Boundary Constrained Floorplanning Using Sequence Pair

Student: Wisely Chen
Advisor: TingTing Hwang

Department of Computer Science
National Tsing Hua University
HsinChu, Taiwan 30043

Contents

List of Figures

List of Tables

Abstract

In this thesis, we will study the boundary constrained floorplanning problem. A floorplan can be classified as slicing or non-slicing structure based on the placement of modules. The floorplan based on non-slicing structure packs modules tighter than floorplan based on slicing structure. Recently, some non-structure representations were proposed. One of these representations is Sequence Pair. Sequence Pair is a very compact representation and can represent all possible floorplan strucutres. In floorplanning, if modules with input/output connections are placed at boundary of the chip, it will save routing area and routing time. Besides, floorplanning is usually done hierarchically in which modules are grouped into different units. It will help if some modules are packed along the boundary of the unit so that they can be put in the neighboring. Therefore, we will focus on the boundary constrained modules placement problem using Sequence Pair representation. First, we will find some rules for Sequence Pair when boundary constraints are given. Based on these rules, we propose a algorithm that will always search solution in legal solution space. In this way, a lot of time is saved due to the pruning of search space. Our algorithm proceeds in two phases: *Pos* and *Neg*. *Pos* permutes the positive sequence using Simulated Annealing and *Neg* permutes the negative sequence exhaustively. At last, we will show our experimental results.

Chapter 1

Introduction

Floorplanning is an important step in physical design of very large scale integration (VLSI) circuits. After the circuit partitioning phase, the area occupied by each block (sub-circuit) can be calculated. In order to complete the layout, we need to arrange the blocks on the layout surface. The arrangement of sub-circuits is done in the floorplan phase [10]. In the floorplan phase, blocks are positioned on a layout surface, while no two blocks are overlapping. The blocks are positioned so as to minimize the total area of the layout. In other words, it packs all the functional modules in a chip without violating design rules to minimize total area and interconnection cost.

A floorplan can be classified as slicing or non-slicing structure based on the placement of modules. Figure 1.1 shows two floorplanning representing slicing and non-slicing structure. The slicing structure reduces the searching space and save time to find solution. Liu and Wong proposed the Polish expression to represent the slicing structure [6]. The Polish expression [6] is one of the efficient slicing structure representations. The non-slicing structure can pack the modules more tightly to save area. Recently, the *Sequence Pair (SP)* [4, 9, 13] and the *Bounded-Sliceline Grid* (BSG) [1, 2, 3] structures were proposed. The advantage of these two methods to the conventional slicing structure [6, 7, 8, 11] is their universality: all placements could be generated, either slicing or non-slicing placement.

In floorplanning, it is useful if IC designers are allowed to specify some placement constraints in the final packing. Murata studied one type of placement constraints: preplaced constraint. In this case, modules with preplaced constraint, like RAM, ROM and central processing unit (CPU) core, are fixed in position, height and width, and other modules are placed in the rest of the chip [5]. The second placement constraint is boundary constraint. This is useful because IC designer may want to place some modules with input/output connections on the boundary of the chip. An example is shown in Figure 1.2 [10]. In this example, since modules 1, 5, 9, 11 are placed on the boundary, these modules have input/output connections on the boundary of the chip, where module 9 and module 11 are placed at right boundary, module 1 is placed at bottom boundary and module 5 is placed at left boundary. It will save the routing area and save the complexity of router in routing step.

The second reason for boundary constrained floorplanner is that floorplanning is usually done hierarchically in which modules are grouped into different units. It will help if some modules are packed along the boundary of the unit so that they can be put in the neighboring at next level of hierarchy. Figure 1.3 shows an example where module 9 in block D connects to module 8 in block C, and module 7 in block C connects to module 2 in block E. We hope that module 9, 8, 7 and 2 will be placed at the boundary of the block D, block C, block C and block E.

Therefore, we will focus on the boundary constrained modules placement problem in this thesis. The modules will be placed using non-slicing structure since non-slicing structure will produce more efficient result in aspect of area than the slicing structure. First, we will find some rules for Sequence Pair when boundary constraints are given. Then, based on the defined legal sequence, our algorithm optimizes area using simulated annealing.

The rest of this thesis is organized as follow. We review previous work on Sequence

Pair in Chapter 2. In Chapter 3, our algorithm is presented. Experimental results will be presented in Chapter 4. Chapter 5 gives concluding remarks.

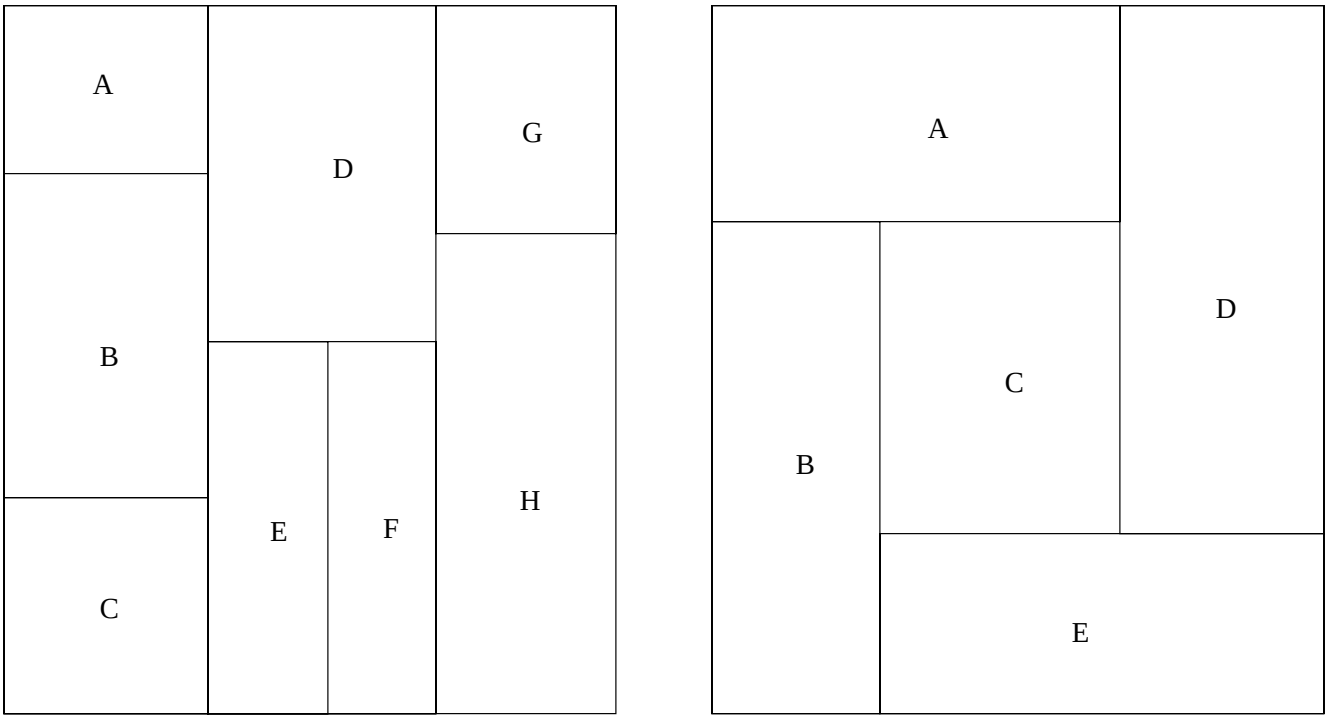


Figure 1.1: Floorplan Structures: (a) Slicing Structure and (b) Non-Slicing Structure

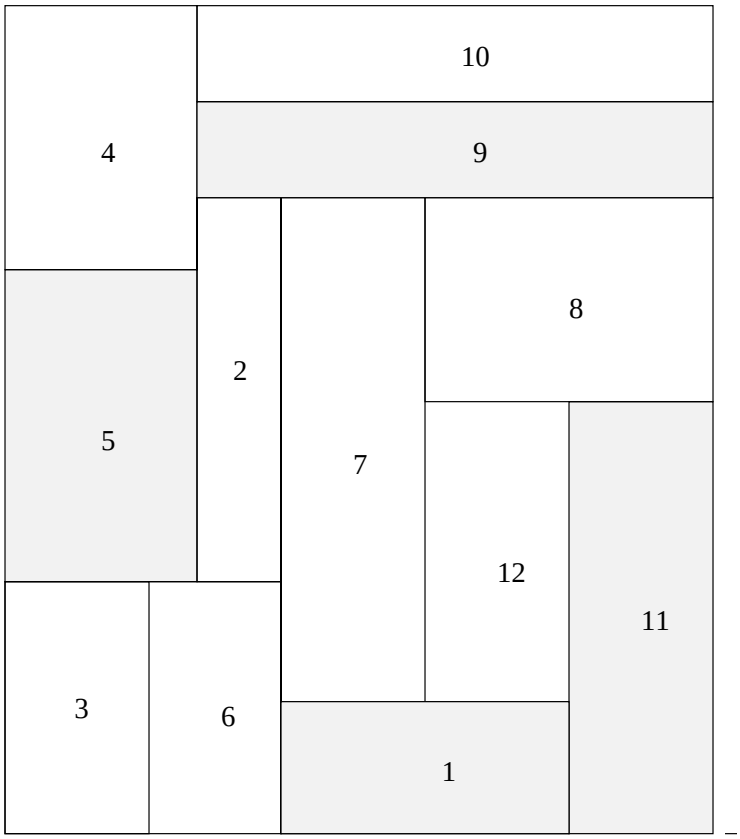


Figure 1.2: Modules with Input/Output Placed on the Boundary of the Chip

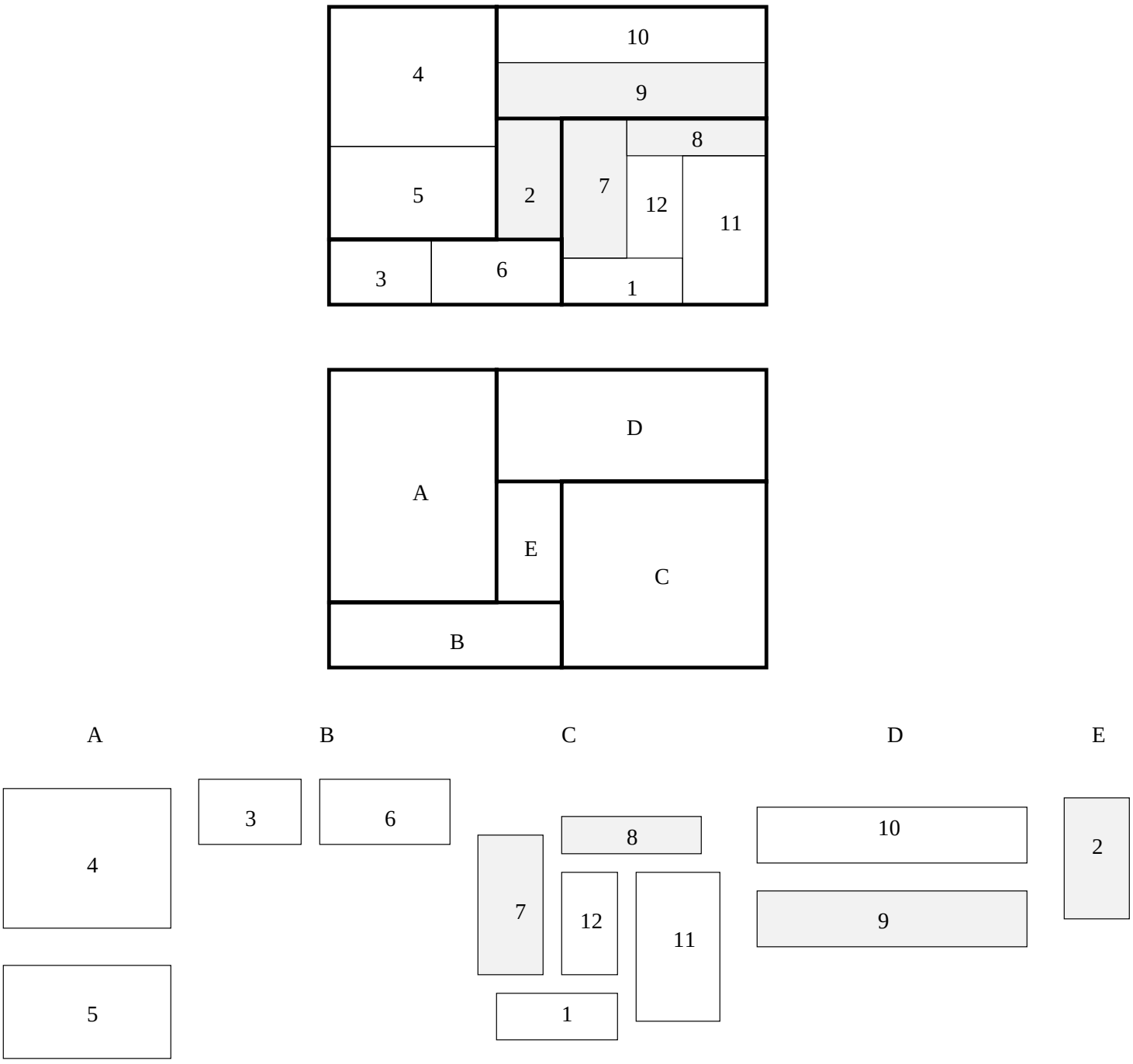


Figure 1.3: Modules Connect to Each Other Between Different Groups by Hierarchical Floorplanning

Chapter 2

Previous Work

2.1 Definition of Floorplanning

Floorplanning is the placement of modules on a two-dimensional plane. The floorplanning problem is defined as follows:

- Given:
  - an electrical circuit consisting of fixed blocks
  - a netlist interconnecting terminals on the periphery of these blocks
  - a netlist interconnecting terminals on the periphery of the circuit itself
- Find:
  - a layout indicating the positions of each block such that the layout area is minimized

A floorplan can be classified as slicing or non-slicing structures. A slicing floorplan is a floorplan which can be obtained by recursively partitioning a rectangle into two parts either by vertical cut or horizontal cut. A slicing structure is the rectangle dissection structure. A slicing structure can be represented by a slicing tree. The

Polish expression is a typical method to represent slicing structure as shown in Figure 2.1. The label $*$ and $+$ are corresponding to vertical cut and horizontal cut.

A non-slicing floorplan is a floorplan that can **NOT** be obtained by recursively partitioning a rectangle into two parts either by vertical cut or horizontal cut. According to this character, a non-slicing floorplanner places modules tighter than slicing floorplanner. The Bounded-Sliceline Grid (BSG) structure was proposed to represent the non-slicing structure. It represents the relative position of each module by using the BSG-table. We can see an simple example in Figure 2.2; Figure 2.2 (a) is a floorplan and Figure 2.2 (b) is its corresponding BSG-table. Disadvantage of BSG structure is that it doesn't cover all possible placements. The Sequence Pair (SP) was also proposed to represent non-slicing structures. It can represent all possible placements. We will discuss in detail in next section.

2.2 Sequence Pair

A Sequence Pair for a set of n modules is a pair of sequences of the modules names. The first sequence is called positive sequence and the second sequence is called negative sequence. For example, *(abc , bac)* is a Sequence Pair for module set *(a, b, c)* where sequence *(abc)* is positive sequence and the sequence *(bac)* is negative sequence.

Given a placement, to derive positive sequence, we start drawing lines for each modules from the left lower corner of the module. The lines will start to move downward and change direction alternatively left and down until it reaches the left lower corner of the chip without crossing. These lines are referred to by the corresponding module names and that is positive sequence. Similarly, we can derive the negative sequence by drawing lines that start from the right lower corner of each module and change direction alternatively right and down until it reaches the right lower corner of the chip without crossing. Figure 2.3 shows how to derive the Sequence Pair for a

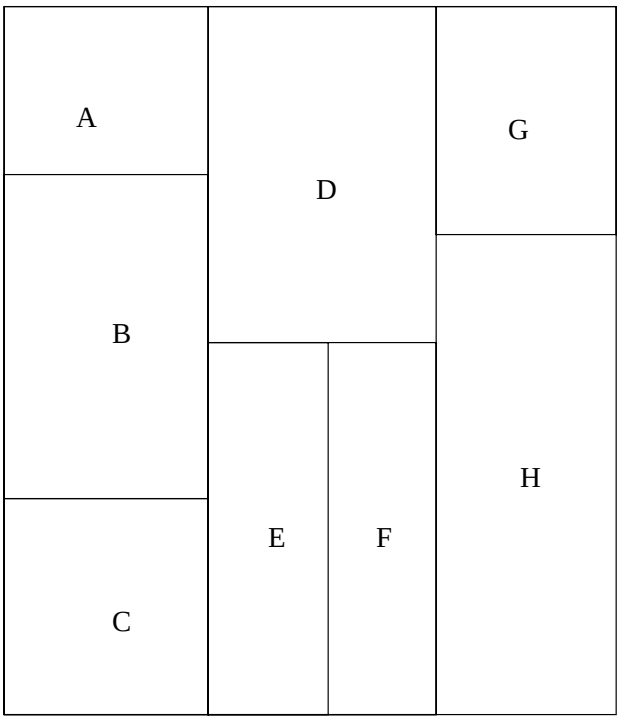
placement. Figure 2.3 (a) and Figure 2.3 (b) show how positive and negative sequence are derived.

A Sequence Pair imposes a horizontal/vertical constraint for every pair of modules as follows:

- *(...a...b... , ...a...b...)* means a should be placed to the left of b
- *(...b...a... , ...a...b...)* means a should be placed below b.

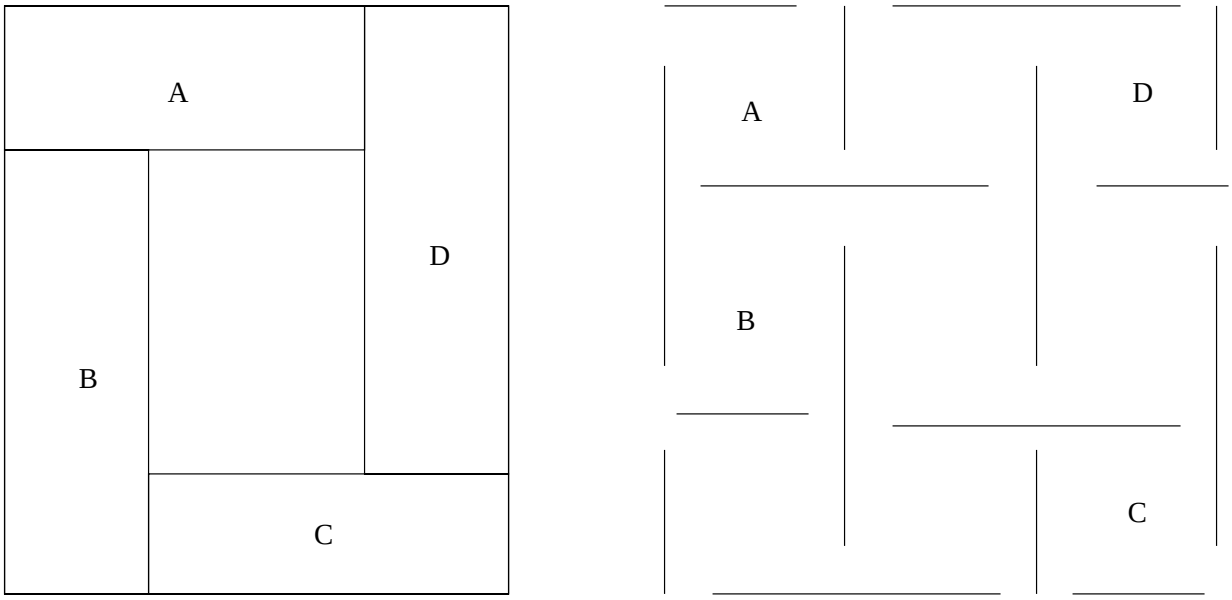
For example, *(abced , bdcae)* means module a, module b and module c should be placed to the left of module c and module a, module c and module e should be placed above module d as shown in Figure 2.4 (a).

The horizontal/vertical constraints of a Sequence Pair can be drawn using the *oblique-grid notation*. The oblique-grid notation of Sequence Pair *(abced , bdcae)* is shown in Figure 2.4 (b). In oblique-grid notation, the nodes acrossed by the same positions in both positive sequence and in negative sequence will form a horizontal line as shown in Figure 2.5 (a). Similarly, the vertical line can be derived in the same way as shown in Figure 2.5 (b). The only difference is that in negative sequence the positions of horizontal line count up from *1* and the positions of vertical line count down from the total number of modules.



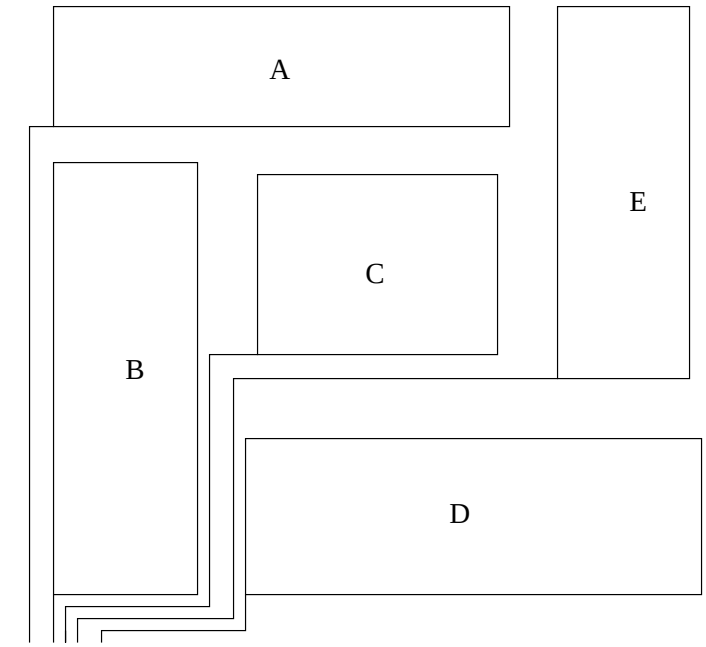
CB+A+EF*D+*HG+*

Figure 2.1: A Slicing Floorplan and Its Corresponding Polish Expression



(a) (b)

Figure 2.2: A Non-Slicing Floorplan and Its Corresponding BSG-table



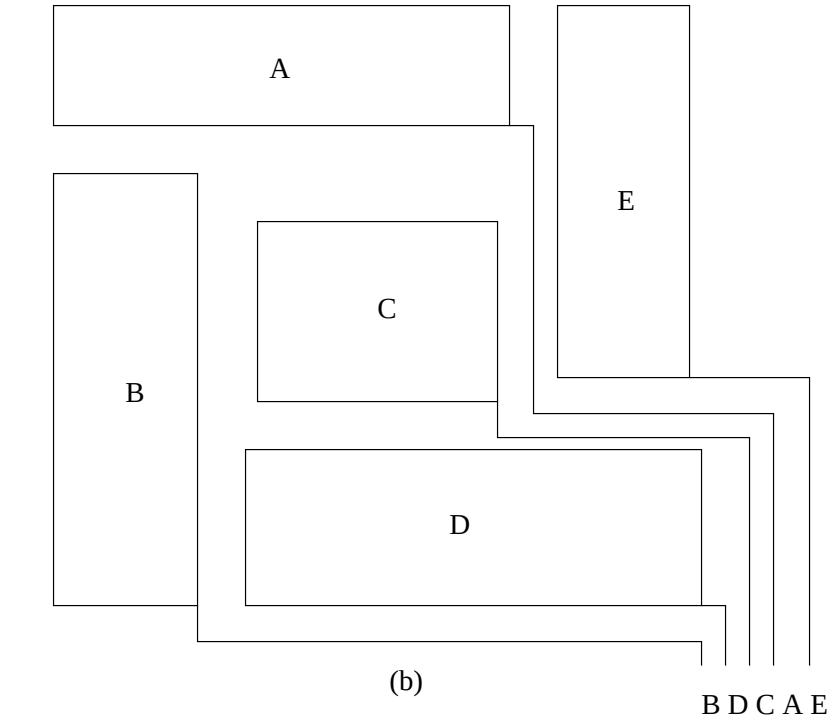
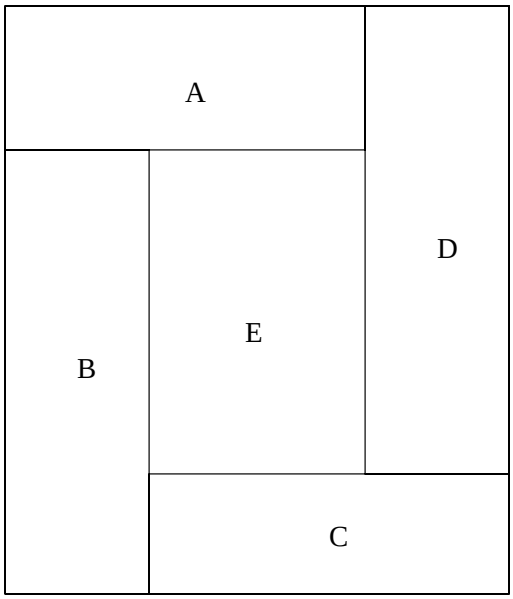
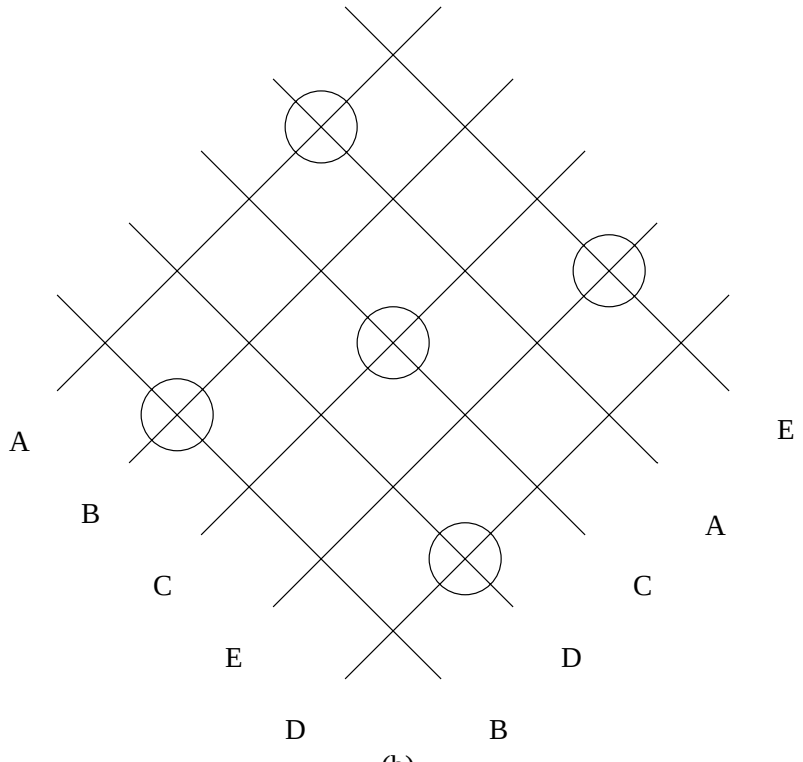


Figure 2.3: A Non-Slicing Floorplan Maps to Sequence Pair: (a)Positive Sequence and (b) Negative Sequence

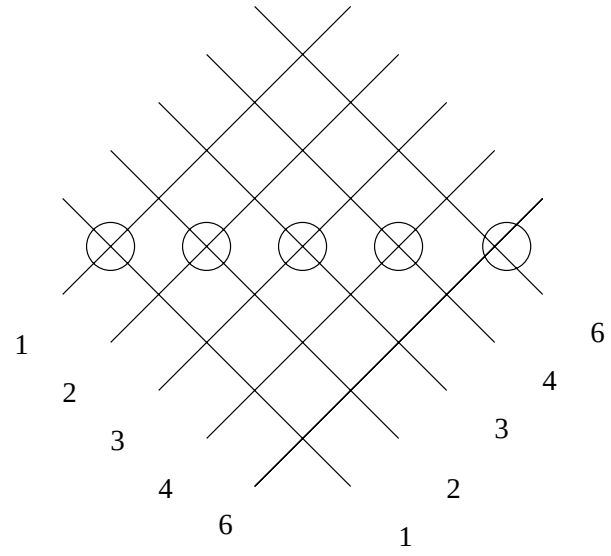


(a)

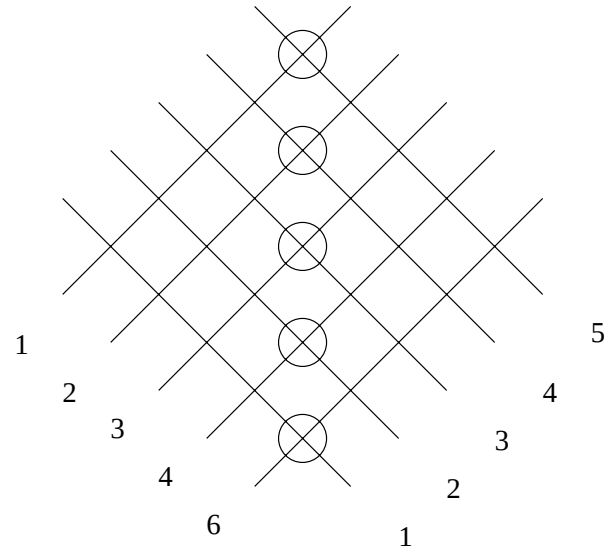


(b)

Figure 2.4: (a)Placement of Sequence Pair *(abced , bdcae)* and (b)Oblique-Grid Notation of Sequence Pair *(abced , bdcae)*



(a)



(b)

Figure 2.5: (a)Horizontal Line (b) Vertical Line

Chapter 3

Proposed Approach

3.1 Boundary Constraints V.S. Sequence Pair

In previous Chapter, we introduce the definition of Sequence Pair where a Sequence Pair contains two parts: positive sequence and negative sequence. In this section, we will find some placement rules in Sequence Pair for the boundary constrained modules.

By definition of Sequence Pair, we know that if module A appears in front of module B both in positive sequence and negative sequence, module A will be placed left to module B. Therefore, module B will never be able to be placed at the left boundary of the final result. Similarly, we can derive rules for modules with top, bottom, left and right boundary constraints.

For top boundary constraint, modules which appear before the top boundary constrained modules in positive sequence must also appear before the top boundary constrained modules in negative sequence. For left boundary constraint, modules which appear before the left boundary constrained modules in positive sequence must appear after the left boundary constrained modules in negative sequence. For bottom boundary constraint, modules which appear after the bottom boundary constrained modules in positive sequence must also appear after the bottom boundary constrained

modules in negative sequence. For right boundary constrained, modules which appear after the right boundary constrained modules in positive sequence must appear before the right boundary constrained modules in negative sequence. The rules for each type of boundary constrained modules are shown as Table 3.1. In Table 3.1, module A denotes the module with boundary constraint and module B without boundary constraint.

Table 3.1: Rules for Boundary Constrained Modules in Sequence Pair.

Boundary Constraint	**Legal permutation**	**Illegal permutation**
Top	$(...a...b..., ...a...b...)$ $(...a...b..., ...b...a...)$ $(...b...a..., ...b...a...)$	$(...b...a..., ...a...b...)$
Left	$(...a...b..., ...a...b...)$ $(...a...b..., ...b...a...)$ $(...b...a..., ...a...b...)$	$(...b...a..., ...b...a...)$
Bottom	$(...a...b..., ...a...b...)$ $(...a...b..., ...b...a...)$ $(...b...a..., ...b...a...)$	$(...a...b..., ...b...a...)$
Right	$(...a...b..., ...b...a...)$ $(...b...a..., ...a...b...)$ $(...b...a..., ...b...a...)$	$(...a...b..., ...a...b...)$

Let T, L, B and R denote the modules with top, left, bottom and right boundary constraints, respectively and F means modules without any boundary constraint. Among these four boundary constraints, there are some rules to be followed. They are presented in the following two lemmas.

Lemma 3.1.1 In positive sequence, there are two rules to be followed.

- T modules must appear in front of R modules
- L modules must appear in front of B modules

$< proof >$ We prove the first rule. Assume the rule is wrong. It means T modules appear behind R modules in the positive sequence. Then, there are two possible permutations due to two permutations for the negative sequence: *(...R...T... , ...R...T...)* and *(...R...T... , ...T...R...)*.

- For permutation *(...R...T... , ...R...T...)*, R will not be able to be placed at the right boundary. This contradicts to our assumption.
- For permutation *(...R...T... , ...T...R...)*, T will not be able to be placed at the top boundary. This contradicts to our assumption.

No matter how the modules are placed in negative sequence, the result is illegal. Therefore, the first rule is proven.

Similarly, we can prove the second rule. In the second rule, if L modules don't appear in front of B modules in positive sequence, L modules or B modules will not be able to be placed at the boundary where they should be placed.

Lemma 3.1.2 In negative sequence, there are two rules to be followed.

- L modules must appear in front of T modules
- B modules must appear in front of R modules

$< proof >$ We prove the first rule. Assume the rule is wrong. It means L modules appear behind T modules in the negative sequence. There are two possible permutations due to the permutations of the positive sequence: *(...L...T... , ...T...L...)* and *(...T...L... , ...T...L...)*.

- For permutation *(...L...T... , ...T...L...)*, T will not be able to be placed at the top boundary. This contradicts to our assumption.

- For permutation *(...T...L... , ...T...L...)*, L will not be able to be placed at the left boundary constraint. This contradicts to our assumption.

No matter how the modules are placed in negative sequence, the result is illegal. Therefore, the first rule is proven.

Similarly, we can prove the second rule. In the second rule, if B modules don't appear in front of R modules in negative sequence, B modules or R modules will not be able to be placed at the boundary where they should be placed.

Rules in Table 3.1, Lemma 3.1.1 and Lemma 3.1.2 only give relative positions of modules when constraints are given. The following Lemma will specify the range of the absolute positions for the constrained modules.

Lemma 3.1.3 Specify the range of the absolute positions for the constrained modules.

- T modules can not be placed below the horizontal line in oblique-grid notation.
- B modules can not be placed above the horizontal line in oblique-grid notation.
- R modules can not be placed left to the vertical line in oblique-grid notation.
- L modules can not be placed right to the vertical line in oblique-grid notation.

$<$ *proof* $>$ We prove the first rule. Suppose one T module, T_a, is placed below the horizontal line. Based on the definition of horizontal line, we know the number of modules which appear before T_a module in positive sequence is larger than the number of modules which appear before module in negative sequence. In other words, there is at least one module that will appear before T_a module in positive sequence and will appear after T_a module in negative sequence. That will cause T module

not to be placed at top boundary by Table 3.1. This contradicts to our assumption. Therefore, the first rule is proven. Similarly, we can prove the other three rules.

For the above rules, we can compute the range of legal placement for boundary constrained modules. Suppose there are N_T number of T modules, N_B number of B modules, N_L number of L modules, N_R number of R modules and N_F number of F modules. The total number of modules is M. P_T denotes last position of T modules and P_L denotes last position of L modules. We can derive the equation for the range of the boundary constraint modules as showns in Table 3.2 and Table 3.3. In Table 3.3, i is the index of the position of the module in the positive sequence and P_i means bound of range in negative sequence of the module that be placed at position i in positive sequence. The legal positions of the constrained modules and free modules for the placement of oblique-grid notation with 2 T modules, 3 L modules, 4 B modules, 5 L modules and 3 F modules are shown in Figure 3.1, Figure 3.2, Figure 3.3, Figure 3.4 and Figure 3.5. We show one example to illustrate how this formula is applied. If T module is placed at fifth position in positive sequence, we can derive the range of position of T module between 5 and 17. Since 5 is larger than N_L (3), we use the equation $P_i = i$ to find lower bound and 5 is larger than N_T, we use the equation $P_i = M$ to find upper bound.

3.2 Algorithm

3.2.1 Range Reduction

Table 3.2 and Table 3.3 give us the ranges of positions of modules in positive sequence and in negative sequence without taking relative positions of modules into consideration. Taking relative position specified by rules as presented in Table 3.1 into account, we can further reduce the range of positions of a module.

Table 3.2: Range for Positive Sequence.

Boundary Constraint	**Lower Bound**	**Upper Bound**
Top	1	$T - N_R$
Left	1	$T - N_B$
Bottom	$P_L + 1$	M
Right	$P_T + 1$	M
Free	1	M

In Table 3.1, we know that given boundary constraints, some modules must appear after other modules in negative sequence. Based on these rules, we can reduce the ranges of position of these modules that must appear after other modules. For example, module a and module b are both L modules. According to Table 3.2 and Table 3.3, module a and module b should have the same range in negative sequence. However, if module a appears before module b in positive sequence, ie. *(...a...b...,.....)*, module a will not appear before module b in negative sequence. Otherwise, it will violate the rule for left boundary constrained modules in Table 3.1. Hence, we can reduce the range of module a. The new range of module a in negative sequence starts at the position where module b is placed.

3.2.2 Proposed Algorithm

Our algorithm proceeds in two phases: *Pos* and *Neg*. *Pos* permutes the positive sequence and *Neg* permutes the negative sequence.

The first step of *Pos* is to randomly pick up legal modules by Table 3.2 to be placed at positions of positive sequence one by one. This selected positive sequence will be a seed for Simulated Annealing.

Before introducing the moves for SA, we define three sets which divide the posi-

Table 3.3: Range for Negative Sequence.

Boundary Constraint	**Lower Bound**	**Upper Bound**
Top	$P_i = N_L + 1$ for $1 <= i <= N_L$ $P_i = i$ for $N_L < i <= M - N_R$	$P_i = M - N_T + i$ for $1 <= i <= N_T$ $P_i = M$ for $N_T <= i <= M - N_R$
Left	$P_i = N_L + 1 - i$ for $1 <= i <= N_L$ $P_i = 1$ for $N_L <= i <= M - N_B$	$P_i = M - N_T$ for $1 <= i <= N_T$ $P_i = M + 1 - i$ for $N_T < i <= M - N_B$
Bottom	$P_i = 1$ for $N_L + 1 <= i <= M - N_B$ $P_i = N_B + i - M$ for $M - N_B < i <= M$	$P_i = i$ for $N_L + 1 <= i <= M - N_R$ $P_i = M - N_R$ for $M - N_R <= i <= M$
Right	$P_i = M + 1 - i$ for $N_T + 1 <= i <= M - N_B$ $P_i = N_B + 1$ for $M - N_B <= i <= M$	$P_i = M$ for $N_T + 1 <= i <= M - N_R$ $P_i = 2M + 1 - i - N_R$ for $M - N_R <= i <= M$
Free	$P_i = T = N_L + 2 - i$ for $1 <= i <= N_L + 1$ $P_i = 1$ for $N_L + 1 <= i <= M - N_B$ $P_i = N_B + 1 + i - M$ for $M - N_B <= i <= M$	$P_i = M - N_T - 1 + i$ for $1 <= i <= N_T + 1$ $P_i = M$ for $N_T + 1 <= i <= M - N_R$ $P_i = 2M - N_R - i$ for $M - N_R <= i <= M$

tions of positive sequence into three sets.

- *The first set* is the positions between the first position and the position where the first modules with right/bottom boundary constraint can be placed.
- *The second set* are the positions that follow the first set and stops at the position where the modules with bottom/right boundary constraint can be placed.
- *The third set* contains the rest of positions.

From Lemma 3.1.1, we know that T modules must appear in front of R modules

and L modules must appear in front of B modules. Therefore, we define the first set ends before the first right boundary constrained module or before the first module with the bottom boundary constraint. Under this condition, only T, L and F modules will appear in the first set. Similarly, the second set only contains either T, B and F modules or L, R and F modules. The third set only contains B, R and F modules.

For example, ($T_j L_l T_k L_m T_n L_o B_p L_q B_r L_s R_t B_u R_v$) is positive sequence. The first set is from T_j to L_o: ($T_j L_l T_k L_m T_n L_o$) ; the second set is from B_p to L_s: ($B_p L_q B_r L_s$) and the third set is from R_t to R_v: ($R_t B_u R_v$).

According to the properties of three sets, we define three moves for SA.

- change the modules with the same boundary constraint
- change the modules in the same set
- – If the last module in the first set is L module and the first module in the second set is R module, change them.
  - If the last module in the first set is T module and the first module in the second set is B module, change them.
  - If the last module in the second set is R module or T module and the first module in the third set is B module, change them.
  - If the last module in the second set is L module or B module and the first module in the third set is R module, change them.

For the first type of move, the modules with the same boundary constraint are allowed to be moved within these module positions. For the second type of move, we can derive all possible permutations in each sets. This move can produce a local optimal solution. For the third type of move, we change the length of three sets. This move can jump away from local optimal permutation. Hence, we can derive the legal global and optimal permutation.

After deciding the positive sequence from *Pos* phase, we can derive the legal placing range in negative sequence for each modules using Table 3.3. According to the information, we can construct a *legal solution table*, where the first column is the positive sequence produced by procedure *Pos* and the first row is the module names. Columns from second column to last column indicate the positions of negative sequence. The elements of *legal solution table* will be set to 0 or 1. When an element is set to 0, it means that the corresponding module can not be placed at that position in negative sequence. Otherwise, the module can be placed at the position. In procedure *Neg*, we will search the best solution exhaustively under the selected positive sequence using these table. The flowchart of proposed algorithm is shown in Figure 3.6.

The objective of our algorithm is to minimize the total area of the chip under the boundary constraints. The are is computed as

$$Area = W \times H \tag{3.1}$$

where W denotes the maximum width of the floorplanning result and H denotes the maximum height of the floorplanning result.

3.3 Example

We illustrate our algorithm using an example. Suppose there are 17 modules, named 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 and 17. 1 and 2 are T modules; 3, 4 and 5 are L modules; 6, 7, 8 and 9 are B modules; 10, 11, 12, 13 and 14 are R modules; 15, 16 and 17 are F modules. The legal position ranges are shown in Figure 3.1 for T modules, Figure 3.3 for L modules, Figure 3.2 for B modules, Figure 3.4 for R modules and Figurer 3.5 for F modules. From Table 3.2 we can calculate a legal positive sequence range for all modules.

At the first position of positive sequence, we know that modules 1, 2, 3, 4, 5, 15, 16 and 17 are acceptable modules to be placed at the position. We randomly choose one module from these modules. Let module 4 be chosen. In the same way, we can choose modules 1, 15, 5 and 3.

After choosing module 3, L modules are all already placed in positive sequence. B modules begin to be candidates. At the sixth position of positive sequence, we know that modules 2, 6, 7, 8, 9, 16 and 17 are acceptable modules to be placed at the position. Repeating the above selection step, we can produce a positive sequence.

Assume that the positive sequence seed produced is (4 1 15 5 3 6 2 14 17 9 10 11 12 8 16 13 7). With this selected positive sequence, the initial negative sequence position range using Table 3.3 is shown in Table 3.4. The range of positions in negative sequence are all set to 1 in Table 3.4. However, there are some illegal positions in the initial negative sequence caused by the selected positive sequence. We will run the range_reduction to remove the illegal position. In Table 3.5, the removed positions are represented by 2. In the first run, module 3 will be placed in the first position in negative sequence, represented by number 3 in the Table 3.6. We will run the range_reduction to remove the illegal positions caused by selecting module 3, represented by number 4 in the Table 3.6. The range_reduction step repeats until all modules are placed. After the range_reduction step, we will search all possible legal permutations in negative sequence and select the best result.

Table 3.4: Position Range in Negative Sequence before Range_Reduction(1 Means Legal Position)

Module Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
4	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
3	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
11	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
12	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
16	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
13	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
7	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0

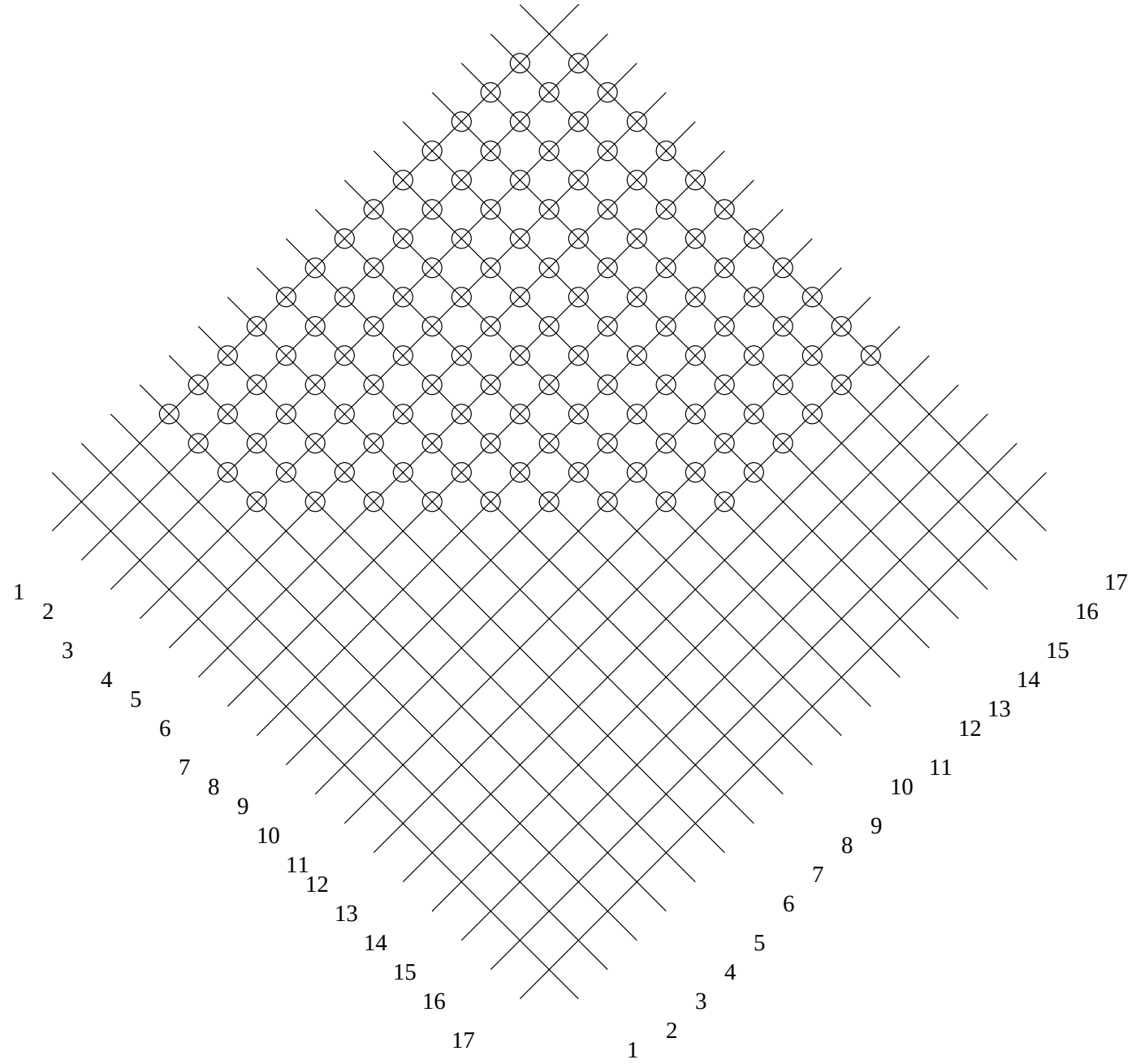


Figure 3.1: Legal Position for Modules with Top Boundary Constraint

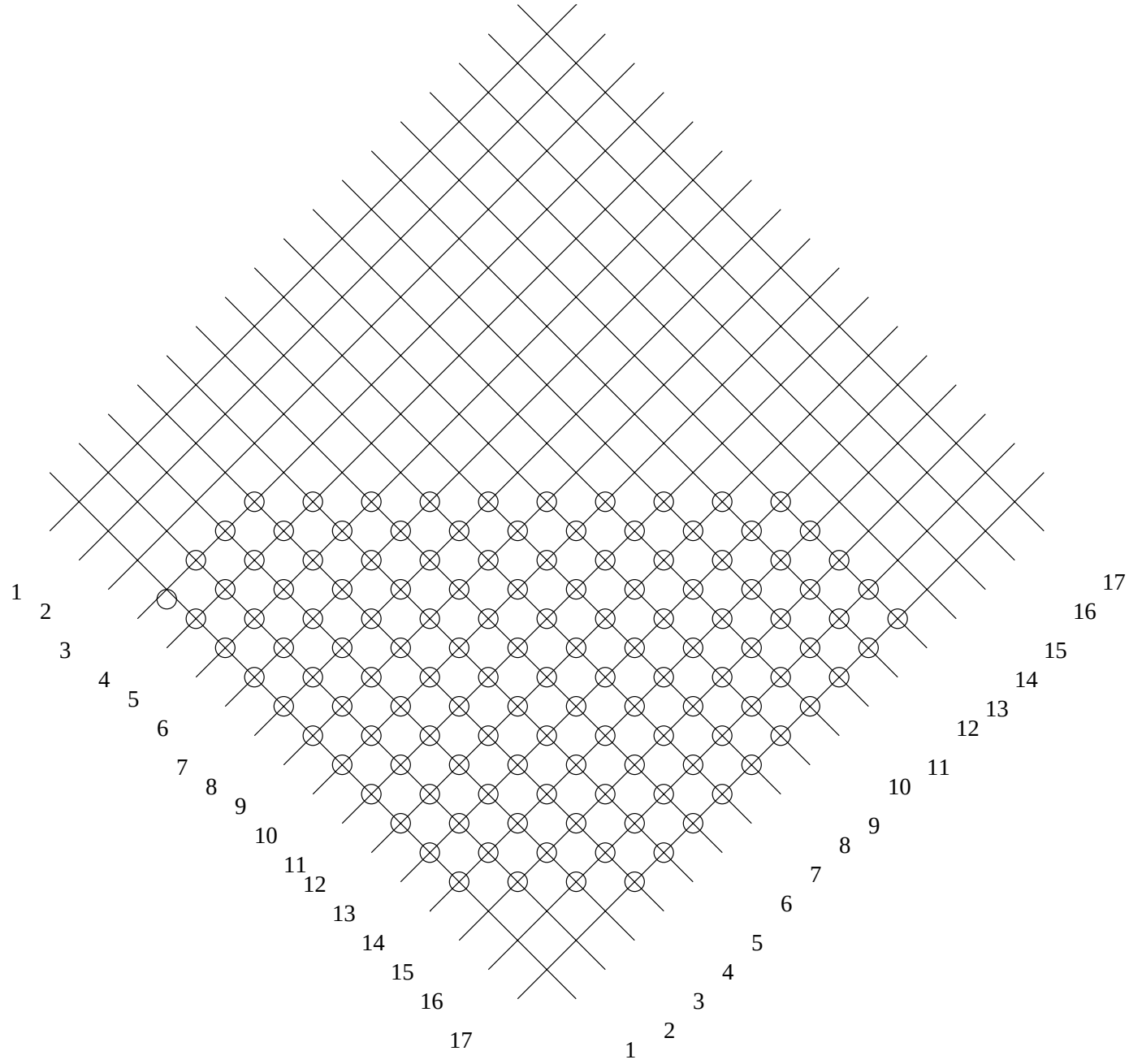


Figure 3.2: Legal Position for Modules with Bottom Boundary Constraint

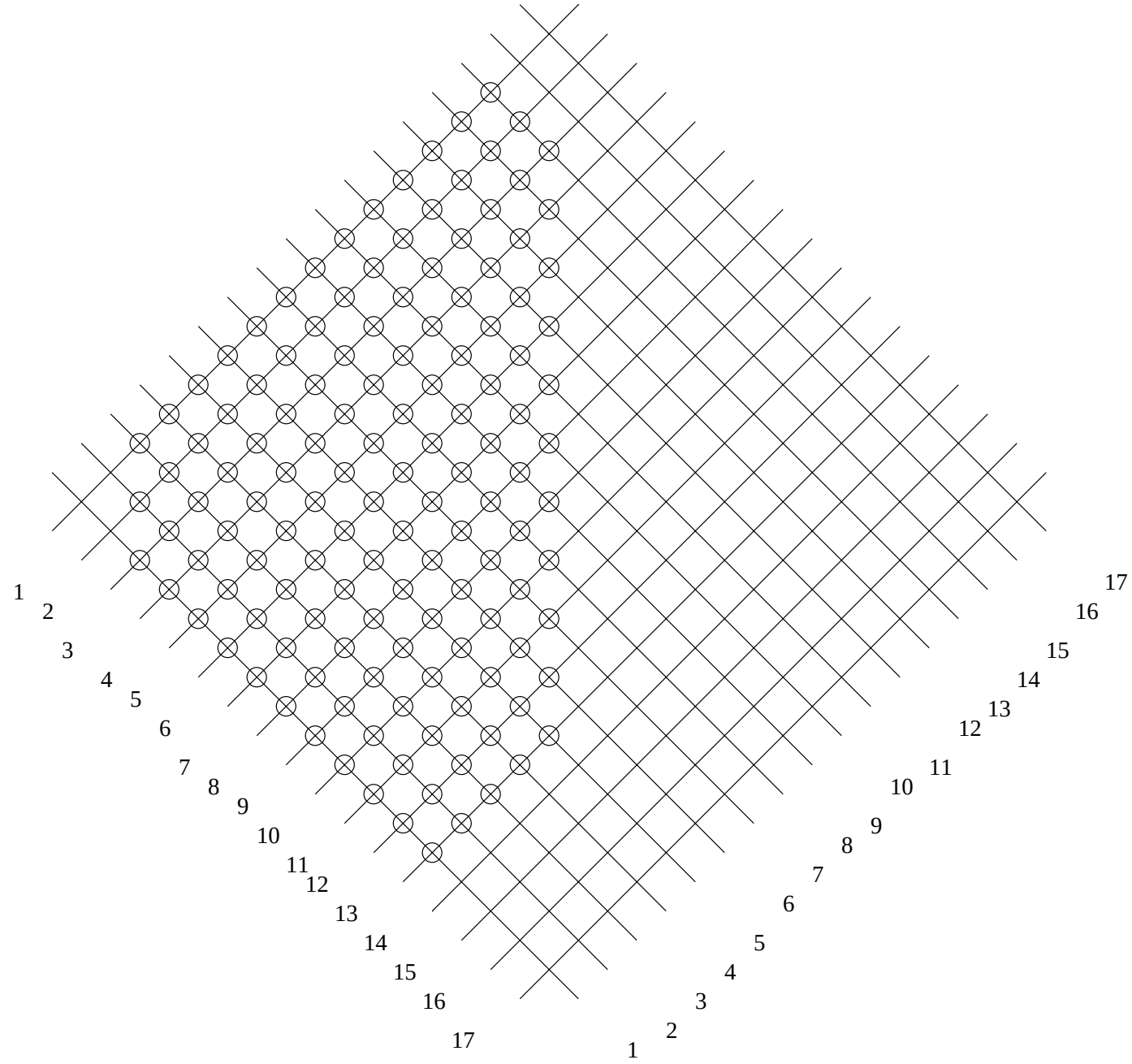


Figure 3.3: Legal Position for Modules with Left Boundary Constraint

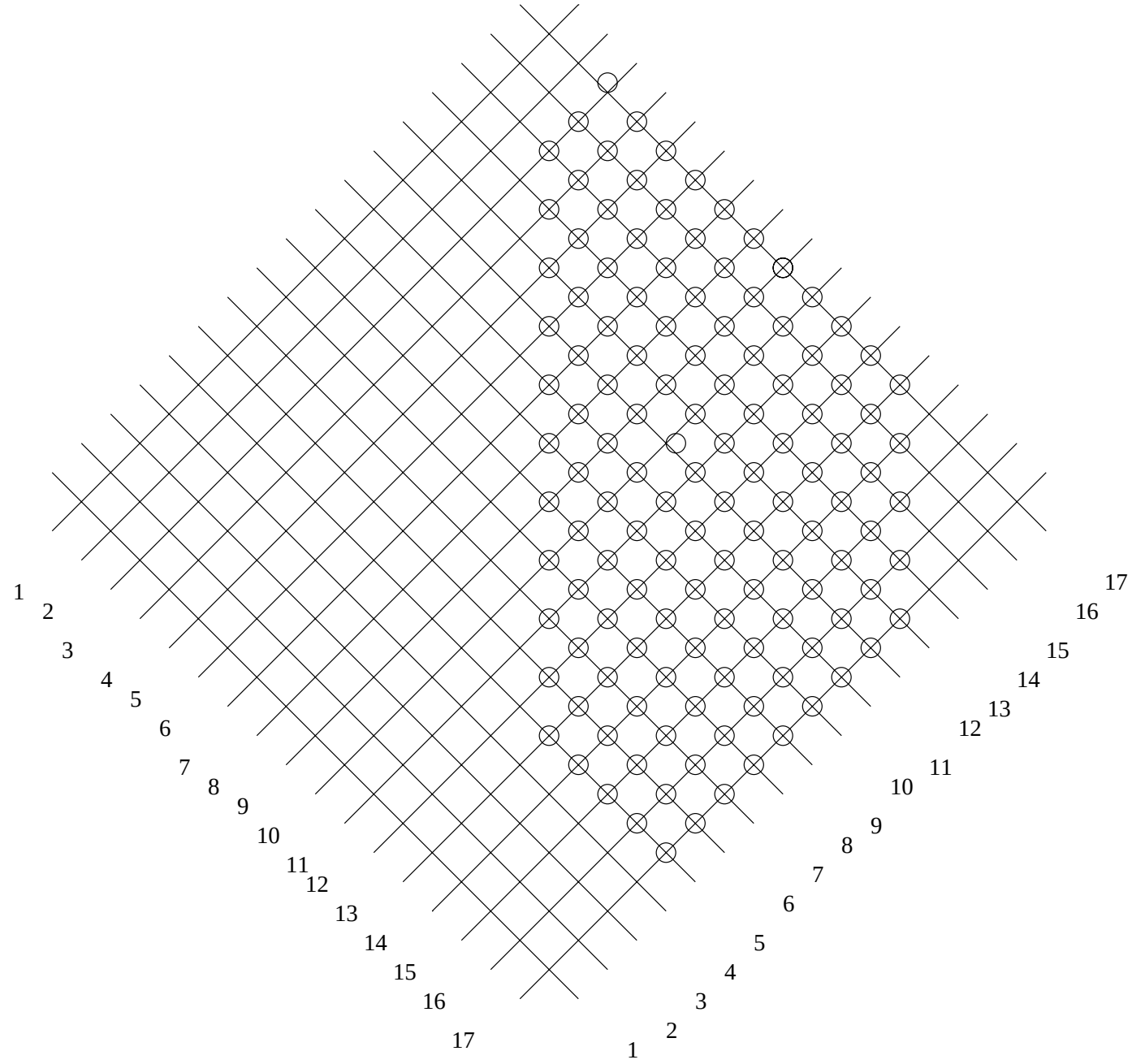


Figure 3.4: Legal Position for Modules with Right Boundary Constraint

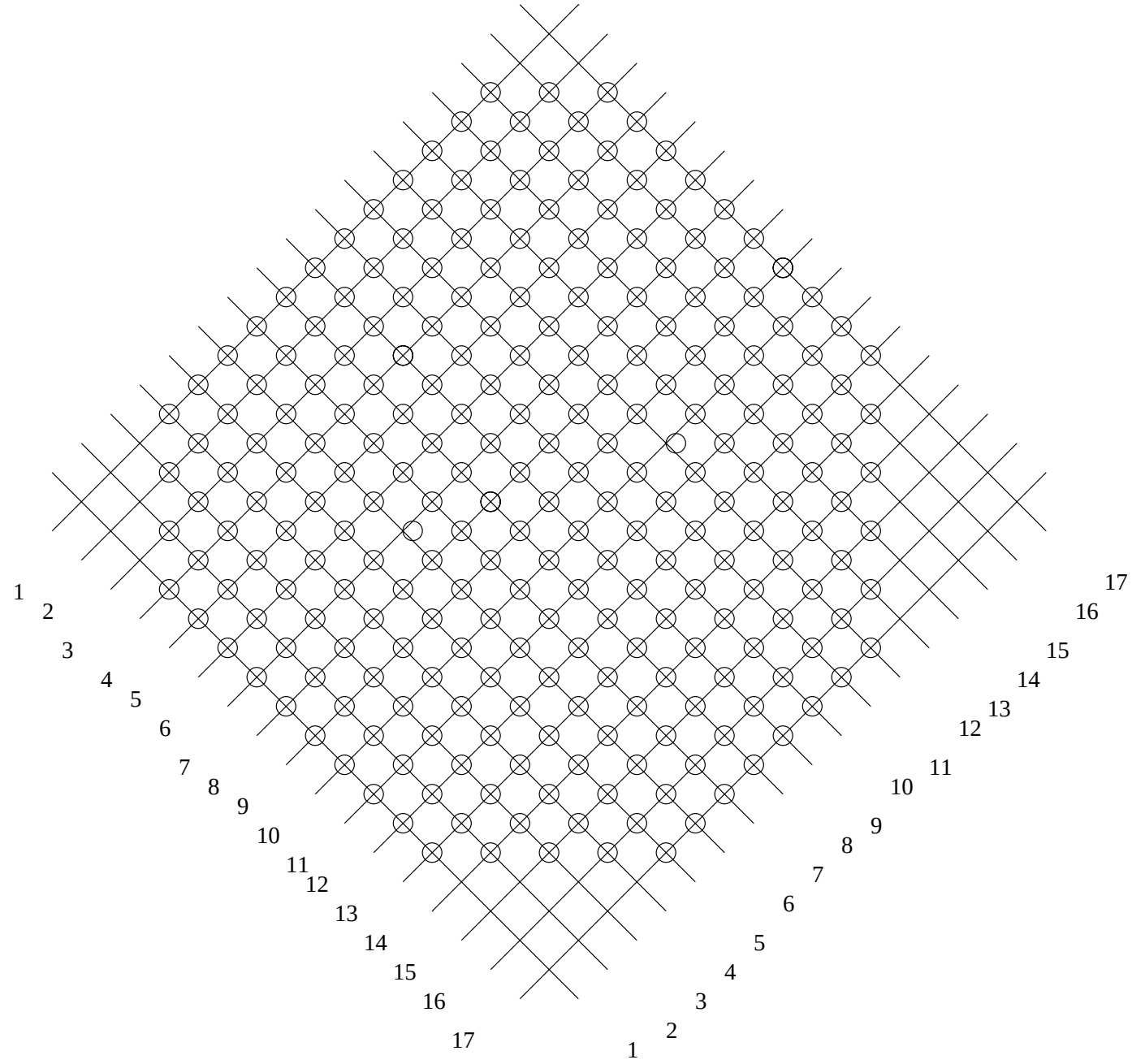


Figure 3.5: Legal Position for Modules without Boundary Constraint

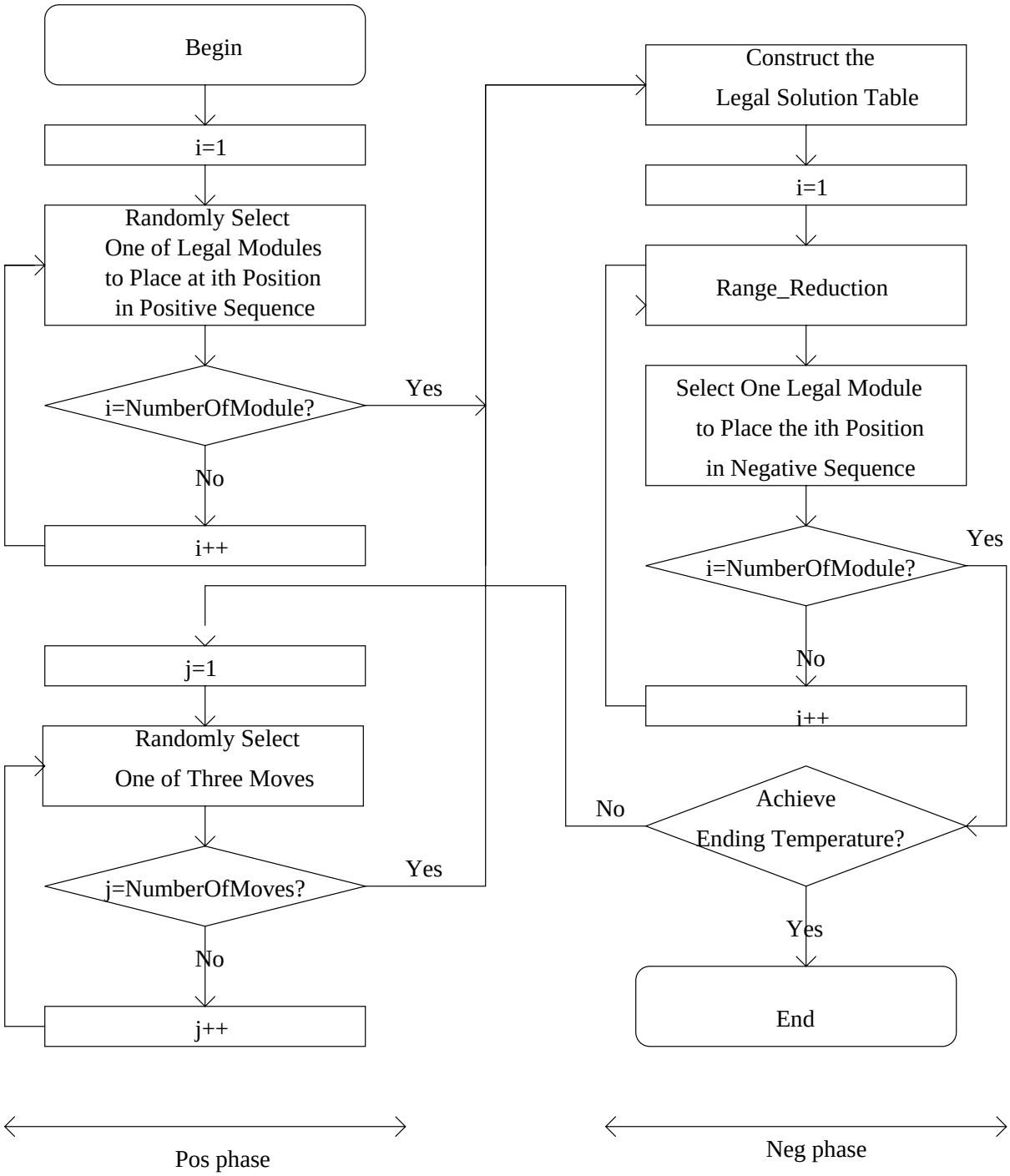


Figure 3.6: Flowchart of Proposed Algorithm

Table 3.5: Position Range in Negative Sequence after Range_Reduction(2 Means Removed by Range_Reduction)

Module Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
4	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2
15	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2
5	2	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
3	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
17	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
9	2	2	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2
11	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2
12	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2
8	2	2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
16	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
13	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
7	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0

Table 3.6: Position Range in Negative Sequence after Selecting Module 3 in First Position (3 Means Selected Module and 4 Means Removed by Range_Reduction)

Module Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
4	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
15	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
5	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
3	3	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
6	4	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
17	0	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
9	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
11	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
12	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0
8	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
16	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
13	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
7	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0

Chapter 4

Experimental Results

The proposed algorithm has been implemented in C language. To test our algorithm, benchmark examples are selected from MCNC. Table 4.1 summarize the characteristics of each circuits. We pick up 9 modules from ami33, 10 modules from ami49, 4 modules from hp, 5 modules from apte and 5 modules from xerox, and require them to be packed along the boundaries evenly as boundary constraints.

Table 4.1: Characteristic of Benchmarks

Benchmark	**Total Area**	**Number of Modules**
Ami33	1168349	33
Ami49	22628920	49
Xerox	19350296	10
Hp	8810760	11
Apte	4651628	9

The starting temperature is 1000K. The temperature is lowered at a constant rate (0.9). All the experimentals were carried out on a 266-MHz Pentium Intel processor.

In our experiments, we compare the modules under two conditions: modules with boundary constraints and modules without boundary constraints. The algorithm of modules without boundary constraints is implemented in dual Simulated Annealing

for positive sequence and negative sequence [4]. In Table 4.2, we show the area utilization. Our result wastes no more than 6% dead space when the boundary constraints are given.

In Table 4.3, we compare two types of placement constraints: preplaced constraint and boundary constraint. In this experiment, we first run our program to get the positions of constrained modules. Then we assign these constrained modules as preplaced modules with fix position in chip and run tool presented in [5]. This Table shows that our algorithm can produce much better results than [5].

Table 4.2: Time and Area of Modules with Boundary Constraint and without Boundary Cnstraint

Bench	**Total Area**	**Area with Dual SA**	**Area in Our Algorithm**	**Time in Our Algorithm**
Ami33	1168349(1)	1285515(1.10)	1350048(1.16)	238032
Ami49	22628920(1)	35581225(1.06)	36537136(1.12)	1023852
Xerox	19350296(1)	20509889(1.06)	37537136(1.09)	2361
Hp	8810760(1)	8926841(1.01)	9419592(1.07)	4473
Apte	4651628(1)	47553329(1.02)	48097872(1.03)	3421

Table 4.3: Area of Modules with Boundary Constraint and Modules with Preplaced Cnstraint

Bench	**Total Area**	**Area with Preplaced Constraint**	**Area with Boundary Constraint**
Ami33	1168349(1)	1609748(1.37)	1350048(1.16)
Ami49	22628920(1)	49670164(2.1)	36537136(1.12)

Chapter 5

Conclusions

In this thesis, will have studied the boundary constrained floorplanning problem. In floorplanning, it is userful if IC designers are allowed to specify some placement constraints in the final packing. In this thesis, we have presented a non-slicing structure floorplanning algorithm,which takes boundary constraint into account. Based on Sequence Pair, we have showed some rules when boundary constraints are given. These rules help us avoid to find illegal permutations in positive sequence and in negative sequence so that we can always find solution from legal solution space when using Sequence Pair.

Bibliography

[1] S.Nakatake, K.Fujiyoshi, H.Murata, and Y, Kajitani, "Module Placement on BSG-Structure and IC Layout Applications," in Proc. of ICCAD, 1996, pp. 484-491.

[2] Jianbang Lai and Ting-Chi Wang, "Module Placement with Boundary Constraint Based on BSG-Structure," in Proc. of Taiwan, 1998, pp. 43-47.

[3] S. Nakatake, M. Furuya and Y. Kajitani, "Module Placement on BSG-Structure with Pre-Placed Modules and Rectilinear Modules,"

[4] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "Rectangle-Packing-Based Module Placement," in Proc. of ICCAD, 1995, pp. 472-479.

[5] H. Murata, K, Fujiyoshi and M. Kaneko, "VLSI/PCB Placement with Obstacles Based on Sequence-Pair," in Proc. of ISPD, 1997, pp. 26-31.

[6] D. F Wong and C. L. Liu, "A New Algorithm for Floorplan Design," in Proc. of DAC, 1986, pp. 101-107.

[7] F. Y. Young and D. F. Wong, "Slicing Floorplans with Boundary Constraint," in Proc. of ASP-DAC, 1997, pp265-270.

[8] F. Y. Young and D. F. Wong, "Slicing floorplans with preplaced modules," in Proc.IEEE Int. Conf. Computer-Aided Design, 1998, pp. 252-258

[9] H. Murata and E. S. Kuh, "Sequence Pair Based Placement Method for Hard/Soft/Pre-Placed Modules", in Proc. of ISPD, 1998, pp. 167-172.

[10] N. Sherwani, "Algorithms for VLSI physical design automation," Kluwer Academic Publishers, 1996.

[11] D. P. Lapotin and S. W. Director, "A new algorithm for floorplan design." Proceedings IEEE International Conference on Computer-Aided Design, page 143-145, 1985.

[12] F. Y. Young and D. F. Wong, "How good are slicing floorplans." Integration, the VLSI journal, 23:61-73, 1997.

[13] Jin Xu, Pei-Ning Guo and Ghung-Kuan Cheng, "Sequence-Pair Approach for Rectilinear Module Placement." IEEE Transactions on Computer-Aided Design, page 484-493, 1999.

[14] Teng-Sheng Moh, Tsu-Shuan Chang and S. Louis Hakimi, "Globally Optimal Floorplanning for a Layout Problem." IEEE Transactions on Circuits and System, page 713-720, 1996.