Interconnect-driven Floorplanning

Sham Chiu Wing

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

©The Chinese University of Hong Kong
June, 2002

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or the whole of the materials in this thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.

香港中文大學
香港中文大學圖書館系統
THE CHINESE UNIVERSITY OF HONG KONG
07 MAR 2003
UNIVERSITY
LIBRARY SYSTEM

電路連接驅動的佈圖規劃

岑超榮

香港中文大學

計算機科學與工程學課程

哲學碩士論文

２００２年６月

摘要

佈圖規劃（floorplanning）在大規模集成電路（VLSI）的設計週期上一向佔有重要的地位。傳統的佈圖規劃器（floorplanner）把重點集中於晶片大小，可是，由於大規模集成電路的發展一日千里，一顆晶片會包含數以百萬計的晶體管，爲要保持晶片的速度，控制電路連接（interconnect）的延時，電路連接的可佈性（routability）和緩衝區的規劃也變得相當重要。

在我的論文中，我們設計了一個電路連接驅動（interconnect-driven）的佈圖規劃器（floorplanner），它會考慮到電路上的電路連接可佈性，在每個位置作電路連接的擁塞估計，務求使到每一條電路連接都能成功分佈於晶片內，並且考慮到緩衝區的需要以及需要加設的位置，這樣便能夠保持訊息在晶片內傳送的速度。實驗證明，在我們的佈圖規劃器的規劃下，可減少最終佈圖規劃（floorplan）上電路連接的擁塞，並且能藉著緩衝區的加設去保持訊息在最終佈圖規劃內傳送的速度。

雖然電路連接驅動的佈圖規劃器能夠有效地減少最終佈圖規劃上電路連接的擁塞，並且能保持訊息在最終佈圖規劃內傳送的速度，但佈圖規劃的大小及佈圖規劃所需的時間亦會因此而略爲增加。所以我們提出了一個尋找替代佈圖規劃（alternative floorplan）的方法。我們發現一個佈圖規劃如果包含一些超級模塊（super-modules），我們可以透過垂直翻轉，水平翻轉或同時垂直及水平翻轉那些超級模塊去找出替代佈圖規劃。實驗證明，佈圖規劃器在使用尋找替代佈圖規劃的方法後，能在很短的時間內更進一步減少最終佈圖規劃上電路線的總長度，同時，佈圖規劃的大小亦能維持不變。

Interconnect-driven Floorplanning

Submitted by

Sham Chiu Wing

for the degree of Master of Philosophy in

Computer Science and Engineering

at the Chinese University of Hong Kong

in June, 2002

Abstract

In traditional floorplanners, area minimization is an important issue. However, due to the recent advances in VLSI technology, the number of transistors in a design are increasing rapidly and so are their switching speeds. This has increased the importance of interconnect delays and routability in the overall performance of a circuit. We should consider interconnect planning, buffer planning and routability as early as possible.

In this thesis, we study and implement two routability-driven floorplanners (based on different assumptions on the buffer requirements) with congestion estimation and buffer planning. The routability-driven floorplanner evaluates the routability of a floorplan by computing the probability that a net will pass through each particular location of a floorplan taken into account buffer locations and routing blockages. Experimental results show that our congestion model can optimize congestion and delay (by successful buffer insertions) of a circuit better with only a slight penalty in area.

Interconnect-driven floorplanners can perform congestion optimization and buffer planning effectively but there is a significant penalty in run time. In

this thesis, we also propose a method to reduce interconnect cost of a floorplan by searching alternative packings. We have found that if a floorplan F contains some rectangular supermodules, we can rearrange those blocks in the supermodule to obtain a new floorplan with the same area as F but with a smaller interconnect cost. Experimental results show that we can reduce the interconnect cost of a floorplan without any penalty in area and runtime by using this method.

Acknowledgments

Special thanks will be given to F. Y. Young.

Contents

1 Introduction **1**

1.1 Motivations 1

1.2 Progress on the Problem 2

1.3 Our Contributions 3

1.4 Thesis Organization 5

2 Preliminaries **6**

2.1 Introduction 6

2.1.1 The Role of Floorplanning 6

2.1.2 Wirelength Estimation 7

2.1.3 Different Types of Floorplan 8

2.2 Representations of Floorplan 10

2.2.1 Polish Expressions 10

2.2.2 Sequence Pair 11

2.2.3 Bounded-Sliceline Grid (BSG) Structure 13

2.2.4 O-Tree 14

2.2.5 B*-Tree 16

2.2.6 Corner Block List 18

2.2.7 Twin Binary Tree 19

2.2.8 Comparisons between Different Representations 20

2.3 Algorithms of Floorplan Design 20

2.3.1 Constraint Based Floorplanning 21
2.3.2 Integer Programming Based Floorplanning 21
2.3.3 Neural Learning Based Floorplanning 22
2.3.4 Rectangular Dualization 22
2.3.5 Simulated Annealing 23
2.3.6 Genetic Algorithm 23
2.4 Summary 24

3 Literature Review on Interconnect-Driven Floorplanning 25
3.1 Introduction 25
3.2 Simulated Annealing Approach 25
3.2.1 "Pepper - A Timing Driven Early Floorplanner" 25
3.2.2 "A Timing Driven Block Placer Based on Sequence Pair Model" 26
3.2.3 "Integrated Floorplanning and Interconnect Planning" . 27
3.2.4 "Interconnect Driven Floorplanning with Fast Global Wiring Planning and Optimization" 27
3.3 Genetic Algorithm Approach 28
3.3.1 "Timing Influenced General-cell Genetic Floorplanning" 28
3.4 Force Directed Approach 29
3.4.1 "Timing Influenced Force Directed Floorplanning" 29
3.5 Congestion Planning 30
3.5.1 "On the Behavior of Congestion Minimization During Placement" 30
3.5.2 "Congestion Minimization During Placement" 31
3.5.3 "Estimating Routing Congestion Using Probabilistic Analysis" 31
3.6 Buffer Planning 32

3.6.1 "Buffer Block Planning for Interconnect Driven Floorplanning" . . . 32
3.6.2 "Routability Driven Repeater Block Planning for Interconnect-centric Floorplanning" . . . 33
3.6.3 "Provably Good Global Buffering Using an Available Block Plan" . . . 34
3.6.4 "Planning Buffer Locations by Network Flows" . . . 34
3.6.5 "A Practical Methodology for Early Buffer and Wire Resource Allocation" . . . 35
3.7 Summary . . . 36

4 Floorplanner with Fixed Buffer Planning [34] 37
4.1 Introduction . . . 37
4.2 Overview of the Floorplanner . . . 38
4.3 Congestion Model . . . 38
4.3.1 Construction of Grid Structure . . . 39
4.3.2 Counting the Number of Routes at a Grid . . . 40
4.3.3 Buffer Location Computation . . . 41
4.3.4 Counting Routes with Blocked Grids . . . 42
4.3.5 Computing the Probability of Net Crossing . . . 43
4.4 Time Complexity . . . 44
4.5 Simulated Annealing . . . 45
4.6 Wirelength Estimation . . . 46
4.6.1 Center-to-center Estimation . . . 47
4.6.2 Corner-to-corner Estimation . . . 47
4.6.3 Intersection-to-intersection Estimation . . . 48
4.7 Multi-pin Nets Handling . . . 49
4.8 Experimental Results . . . 50
4.9 Summary . . . 51

5 Floorplanner with Flexible Buffer Planning [35] 53

5.1 Introduction 53

5.2 Overview of the Floorplanner 54

5.3 Congestion Model 55

5.3.1 Probabilistic Model with Variable Interval Buffer Insertion Constraint 57

5.3.2 Time Complexity 61

5.4 Buffer Planning 62

5.4.1 Estimation of Buffer Usage 62

5.4.2 Estimation of Buffer Resources 69

5.5 Two-phases Simulated Annealing 70

5.6 Wirelength Estimation 72

5.7 Multi-pin Nets Handling 73

5.8 Experimental Results 73

5.9 Remarks 76

5.10 Summary 76

6 Global Router 77

6.1 Introduction 77

6.2 Overview of the Global Router 77

6.3 Buffer Insertion Constraint and Congestion Constraint 78

6.4 Multi-pin Nets Handling 79

6.5 Routing Methodology 79

6.6 Implementation 80

6.7 Summary 86

7 Interconnect-Driven Floorplanning by Alternative Packings 87

7.1 Introduction 87

7.2 Overview of the Method 87

7.3 Searching Alternative Packings 89

7.3.1 Rectangular Supermodules in Sequence Pair 89
7.3.2 Finding rearrangable module sets 90
7.3.3 Alternative Sequence Pairs 94
7.4 Implementation 97
7.4.1 Re-calculation of Interconnect Cost 98
7.4.2 Cost Function 101
7.4.3 Time Complexity 101
7.5 Experimental Results 101
7.6 Summary 103

8 Conclusion **105**

Bibliography **107**

List of Figures

2.1 Physical design cycle 7
2.2 Deadspace of the layout 8
2.3 Estimation of wirelength 8
2.4 Relationship between non-slicing, mosaic and slicing floorplan . 9
2.5 Example for different floorplans 9
2.6 Examples of Polish Expressions 11
2.7 Corresponding packing of a Sequence Pair 11
2.8 Horizontal and vertical constraint graph for a SP 13
2.9 Structure of BSG 14
2.10 Constructing constraint graphs from a BSG 15
2.11 The O-Tree and its representation string 16
2.12 An O-tree and its corresponding block placement 17
2.13 A B*-Tree and its corresponding block placement 17
2.14 Example of transformation from floorplan to Corner Block List . 18
2.15 Building a pair of twin binary trees from a mosaic floorplan . . 19
2.16 Simulated Annealing 23

3.1 Algorithm of buffer block planning 33
3.2 Piecewise-linear congestion cost function 34

4.1 Division of floorplan 39
4.2 Examples of blocked grids 40
4.3 Example for counting the number of routes for a net 41

4.4 Example of blocked grid overlaps with the location of buffer insertion 43
4.5 Results of $g_1(x, y, k)$ and $g_2(x, y, k)$ with blocked grids 43
4.6 Example of calculating $P(x, y, k)$ for net k at grid (x, y) 44
4.7 Simulated Annealing 45
4.8 Center-to-center estimation 47
4.9 Corner-to-corner estimation 48
4.10 Intersection-to-intersection estimation 49
4.11 Distribution of wires among the grids for ami33 50
4.12 Distribution of wires among the grids for ami49 51
4.13 Distribution of wires among the grids for playout 52

5.1 A commonly used congestion model 55
5.2 A simple example of our congestion model 56
5.3 Example of computing $route_success(l)$ 58
5.4 Result of $g_0(x, y, k).inf[i]$'s 60
5.5 Result of $g_1(x, y, k).inf[i]$'s 60
5.6 Example of calculating the probabilities of successful buffer insertion at each grid 63
5.7 The algorithm of Forward recursive method 65
5.8 Example of the forward recursive method 66
5.9 The algorithm of backward recursive method 67
5.10 Backward recursive method 67
5.11 The algorithm of combined method 69
5.12 Simulated Annealing 71
5.13 Intersection-to-intersection estimation 72

6.1 Construction of table with information of congestion and empty space 79
6.2 Updated information of congestion and empty space 80

6.3 Different possible routes 81
6.4 Global Router 82
6.5 Procedures of finding routes 83
6.6 Example of the algorithm 85

7.1 Example of alternative packings 88
7.2 The algorithm of FRSS 92
7.3 An example of finding rearrangable module sets by FRSS 93
7.4 Example of alternative sequence pairs 94
7.5 Vertical flip 95
7.6 Horizontal flip 96
7.7 Diagonal flip 96
7.8 Example of identical alternative packing 97
7.9 Re-calculation of the positions of the modules in the new packing 99
7.10 Difference between the actual positions of the module and the estimated positions 100

List of Tables

2.1 Comparisons of different floorplan representations on solution space 20

2.2 Comparisons of different floorplan representations on flexibility and efficiency 21

3.1 Probability Matrix 33

4.1 Configuration of experiment 42

4.2 Comparison between the single level floorplanner with and without congestion estimation 52

5.1 Configuration of the experiments 73

5.2 Comparison between $F1$ and $F3$ 74

5.3 Comparison between $F3$ and $F2$ 75

7.1 Information of data sets 102

7.2 Maximum and average improvement on wirelength by applying the method of searching alternative packings on the final solution floorplan 103

7.3 Comparisons between $F1$ and $F2$ 104

Chapter 1

Introduction

1.1 Motivations

Floorplanning plays an important role in physical design of VLSI circuits. It plans the shapes and locations of the modules on a chip, and the result of which will greatly affect the overall performance of the final circuit. In the past, area minimization is the major concern in floorplan design. As technology continues to scale down, the sizes of the transistors are getting smaller and a significant portion of the circuit delay is coming from interconnects. In some advanced systems today, as much as 80% of the clock cycle is consumed by interconnects [5]. The domination of interconnect in system performance has made area minimization less important while routability and delay has become the major concern in floorplanning and many other designing steps.

Traditional floorplanners have not paid enough attention to interconnect optimization. This results in a large expansion in area, or even an unroutable design failing in achieving timing closure after detailed routing. Buffer insertion is one of the most popular and effective techniques to achieve timing closure. It was projected that over 700K repeaters will be inserted on a single chip for the 70 nm technology [6]. In the current practice, buffers are inserted after routing. However, buffers also take up silicon resources (40× to 200× minimum inverter size) and cannot be inserted wherever we want. Thus,

buffers should be planned in the early stage so that the rest of the design process may make better of use the buffer resources. Besides, buffers contribute delay and area, and their locations should be carefully planned. A good planning of the module positions during the floorplanning stage so that buffers can be inserted wherever needed in the later routing stages will be very useful.

In addition, the congestion problem is also very important. An excessive congestion will result in a local shortage of routing resource. It will lead to a large expansion in area, or even an unroutable design failing in achieving timing closure after detailed routing. In the floorplanning stage, accurate routing demand is not available but it is too late to make significant changes in placement after detailed routing. Thus, a good estimation algorithm is required for congestion analysis during the earlier stages of the design process.

1.2 Progress on the Problem

There are several previous works addressing the interconnect issues in floorplan design. In the papers [31, 13, 48, 5, 3], the authors have formulated different congestion-related cost functions (always evaluated by some simple global routing) including a hybrid length plus congestion and these cost functions are then optimized by applying some heuristics such as simulated annealing and genetic algorithm. In the paper [5], a floorplan is divided into grids and congestion is estimated at each grid, assuming that each wire is routed in either L-shaped or Z-shaped. They use a simple and fast congestion model but buffer insertion is not considered. The paper [42] have found that the congestion-related cost functions cannot reduce congestion effectively. It is because simple global routing may have a large difference with the final detail routing. Thus, they proposed a post processing stage to minimize congestion by applying a net-centric algorithm. Besides that, Lou et al. [20] apply probabilistic analysis to

estimate congestion and routability, and they show that their estimations correlate well with post-route congestion. However their congestion model does not take into account buffer insertions.

Cong et al. define in their paper [7] the term *feasible region* of a net, which is, the largest polygon in which a buffer can be inserted such that the timing constraint can be satisfied. The locations of these feasible regions can be computed and buffers are clustered into blocks in these feasible regions along the channel areas. Sarkar et al. [33] adds into the notion of independence to feasible regions so that the the feasible regions of different buffers on a net can be computed independently. They also try to improve routability by considering congestion in their objective function. Tang and Wong [38] proposed an optimal algorithm to assign buffers to buffer blocks assuming that only one buffer is needed per net. In the paper [41], a realistic global router is used to evaluate congestion of each candidate placement solution. Dragan et al. [9] use a multi-commodity flow-based approach to allocate buffers to some pre-existing buffer blocks such that the required upper and lower bounds on buffer intervals can be satisfied as much as possible. Alpert et al. [2] make use of tile graph and dynamic programming to perform buffer block planning. They propose that buffers should be allowed to be inserted inside macro blocks and their method will distribute buffer sites throughout the layout.

1.3 Our Contributions

In order to address the interconnect issues in floorplan design, we propose three different approaches to achieve the goal.

First, we propose to use a probabilistic method to estimate congestion and routability with buffer insertions [34]. Probabilistic analysis can address the problem in a more global way and can thus give a more accurate estimation on the interconnect information. In order to consider the buffer issues in

congestion estimation, we need to compute the exact buffer locations for each net. Dynamic programming [8] is used to compute the buffer locations in an interconnect tree in the paper [17]. In order to speed up the process, we employ a table look-up approach to store and retrieve the delay and buffer information of a net. This allows flexible handling of the buffer information efficiently.

Second, we propose another probabilistic method to estimate congestion and routability under a more realistic assumption on buffer locations [35]. Instead of computing the exact locations of the buffers, we assume that buffers are constrained to be inserted for long enough wires such that the distance between adjacent buffers is lying within a range $[low, up]$ given by the user. We call this the *variable interval buffer insertion constraint* [9, 2]. For example, according to [14], global repeater rules for a high-end microprocessor design in 0.25 μm CMOS requires repeaters at intervals of at most 4500 μm, and we can model this situation by assigning the *low* and *up* appropriately. In this floorplanner, we will estimate congestion at each grid subjecting to the given buffer insertion constraint. Notice that the congestion at each location is dependent on the routes of the wires while the route of a wire is, in turn, dependent on the availability of buffer resources along the route. This availability of buffer resources can be estimated from the amount of empty space at each location and the number of possible buffer insertions at that location. We compute the congestion information assuming that every route is equally likely to be used as long as the buffer insertion constraint can be satisfied. The computation can be performed efficiently by using dynamic programming and a table lookup approach. Besides, we have used a two stage simulated annealing method to speed up the whole floorplanning process. In order to verify the usefulness of our method, we have implemented a simple global router for testing and the experimental results can demonstrate the effectiveness and efficiency of our floorplanner.

Finally, we propose a method to reduce the interconnect cost of a floorplan

by searching alternative packings. We have found that if there is a floorplan F containing some rectangular supermodules, we can rearrange the blocks in the supermodules to obtain a new floorplan with the same area as F but with a smaller interconnect cost. In general, the alternative packings can be obtained by flipping the blocks in some rectangular supermodules either horizontally or vertically or both horizontally and vertically. Experiments show that we can reduce the interconnect cost of a given floorplan without any penalty in area and runtime by using this method.

1.4 Thesis Organization

This thesis is organized as follows. We will give an introduction in chapter 1. There will be an overview on flooplanning in chapter 2. Some papers on interconnect-driven floorplanning will be reviewed in chapter 3. Floorplanner with fixed buffer planning and flexible buffer planning will be discussed in chapter 4 and chapter 5 respectively. The simple global router that is used to evaluate the floorplanners will be discussed in chapter 6. The interconnect-driven floorplanner will be discussed in chapter 7. Finally, we will have a conclusion in chapter 8.

Chapter 2

Preliminaries

2.1 Introduction

Floorplanning is an important step in the physical design cycle of VLSI circuits. As circuits become very large, an automatic block placement tool is needed to satisfy both the timing and area constraints. In traditional floorplanners, area minimization is an important issue. However, due to the advances in VLSI technology, sizes of transistors have been decreasing and their switching speeds increasing. This has increased the importance of interconnect delay with respect to the overall performance of the circuit. In this chapter, we will have a review on floorplanning.

2.1.1 The Role of Floorplanning

VLSI circuit design is an extremely complex process today. A design may have billion of transistors and it is difficult to handle them manually. In order to achieve a design with good performance, several stages such as partitioning, floorplanning, placement, routing and compaction are included in the physical design cycle. An example is shown in figure 2.1.

After partitioning, the whole circuit is divided into many sub-circuits or blocks. At that time, the area of each block can be estimated. In the floorplanning stage, the floorplanner can minimize the deadspace between the blocks

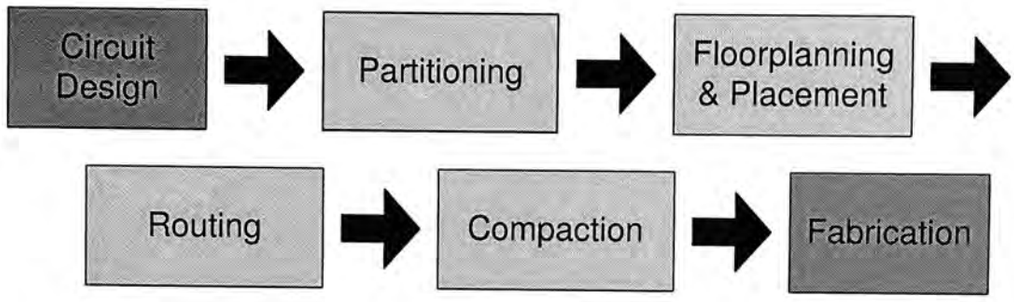


Figure 2.1: Physical design cycle

and the total wirelength of the interconnections. As a result, floorplan design is one of the most important problems in VLSI circuit layout [27].

A poor floorplan design will result in a packing with a lot of deadspace (figure 2.2). The deadspace is the space that is wasted. A floorplanner will try to minimize the total area of the circuit. In addition, the floorplanner will also try to minimize the total wirelength of the packing. It is because longer wirelength may result in larger time delay. Although the exact wirelength of a circuit cannot be known until routing is performed because the exact pin locations are unknown in the floorplanning stage, some appropriate methods for wirelength estimation will be applied to achieve the goal. Moreover, the final circuit layout should be routable. Some appropriate congestion models have been proposed to model the routability of the packing in order to minimize the congestion of the packing during floorplanning stage. Thus, there are three major objectives in floorplanning: area optimization, time delay optimization and routability.

2.1.2 Wirelength Estimation

In the floorplanning stage, the positions of the I/O pins in the modules are not yet fixed so we cannot calculate the actual wirelength in this stage. In traditional floorplanner, half-perimeter estimation and center-to-center estimation using the center of a module as the location of the I/O pins are most commonly

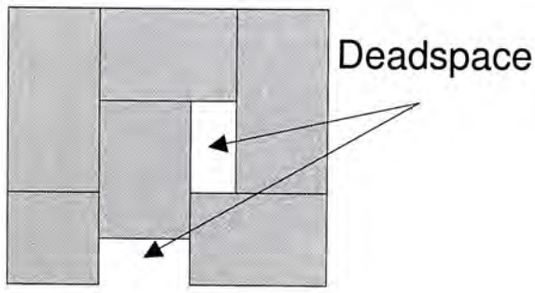


Figure 2.2: Deadspace of the layout

used. An example is shown on figure 2.3.

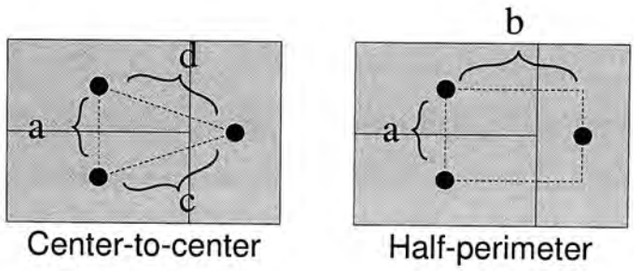


Figure 2.3: Estimation of wirelength

From the example of figure 2.3, if we use half-perimeter estimation, the wirelength is equal to $a + b$. In general, the wirelength is equal to the half-perimeter of the smallest rectangle that can bound inside all the centers of the modules. If we use center-to-center estimation, the wirelength is equal to $a + c + d$. It is equal to the summation of the distances between the centers of the modules.

2.1.3 Different Types of Floorplan

There are three types of floorplan: slicing, mosaic and general non-slicing floorplan (figure 2.4).

A slicing floorplan is one that can be obtained by recursively cutting a rectangle into two parts by either a vertical or a horizontal line [43]. Otherwise,

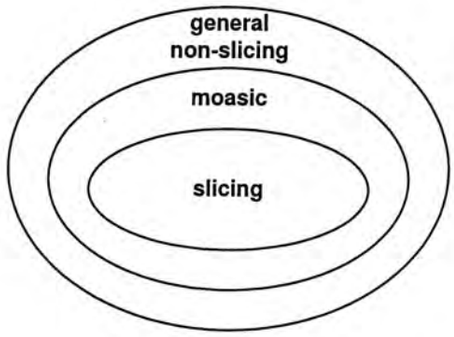


Figure 2.4: Relationship between non-slicing, mosaic and slicing floorplan

it is a non-slicing floorplan. A mosaic floorplan, first introduced in paper [12], is similar to a general non-slicing floorplan except that there is no empty room within the floorplan, i.e., each rectangle is assigned to one and only one block. Consequently, the general non-slicing floorplans can represent all packings including slicing and mosaic floorplans. An example is shown in figure 2.5.

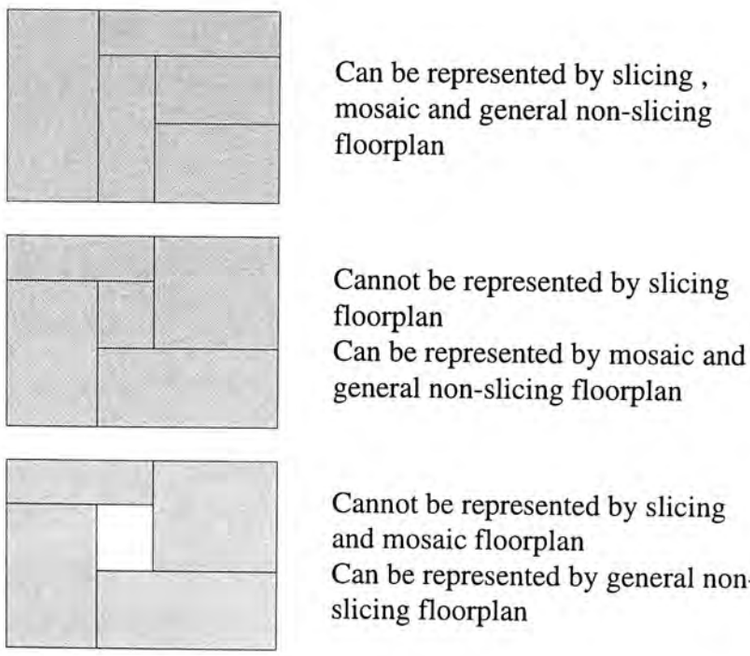


Figure 2.5: Example for different floorplans

In general, slicing floorplan is computationally easier to handle [28] but its application is very limited since most of the dissections are non-slicing [15]. Similarly, mosaic floorplan is computationally easier to handle than general non-slicing floorplan but it cannot represent all floorplans.

2.2 Representations of Floorplan

Floorplan can be described by different representations. In this section, different floorplanning representations are shown.

2.2.1 Polish Expressions

Slicing structures can be represented by either series-parallel graphs [28] or slicing trees [29]. Polish expression is an inorder traversal of the slicing tree such that each internal node of the tree is labeled either by a $*$ or a $+$, corresponding to either a vertical or a horizontal cut, respectively. Each leaf corresponds to a basic rectangle and is labeled by a number between 1 and n when the slicing structure has n basic rectangles [43]. An example is shown in figure 2.6.

However, there may be two or more Polish expressions (slicing trees) that correspond to a given slicing structure. This makes Polish expressions an undesirable choice for representing a solution in a simulated annealing process. As a result, normalized Polish expression is used. The normalized Polish expression is a Polish expression that no consecutive *'s or +'s appear in the Polish expression [43].

Polish expression is efficient and flexible to deal with hard, pre-placed, soft and rectilinear modules. It has a small construction time and a packing can be obtained from the slicing tree directly, i.e., no transformation process from the tree to the constraint graphs is needed. In addition, it can evaluate the area cost incrementally. However, it can handle slicing structures only.

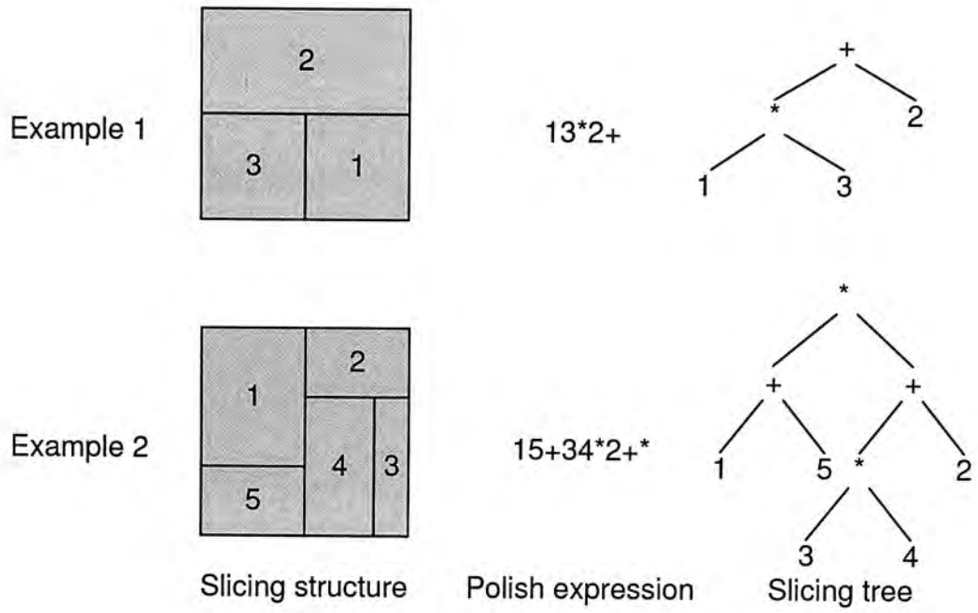


Figure 2.6: Examples of Polish Expressions

2.2.2 Sequence Pair

Sequence Pair (SP) [21] is a very simply and powerful representation of a packing of rectangular modules [22]. It can represent any general non-slicing structure and every sequence pair will correspond to a feasible packing. An example is shown in figure 2.7.

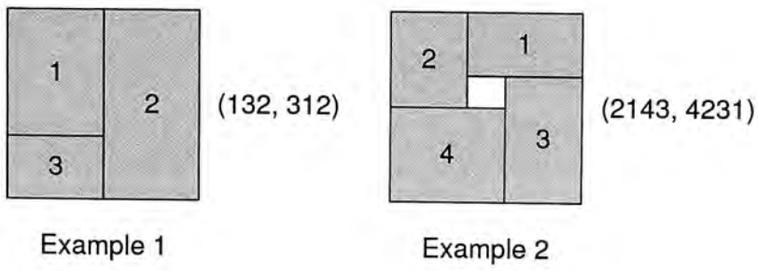


Figure 2.7: Corresponding packing of a Sequence Pair

A Sequence Pair consists of two ordered sequences (S_1, S_2),

S_1: $(s_{1,1}, s_{1,2}, ...s_{1,m})$ and S_2: $(s_{2,1}, s_{2,2}, ...s_{2,m})$,

where the sequence elements $s_{i,j}$ are unique integers from $0, 1, ...M-1$. These integers are identifiers of the modules in the placement problem. From S_1 and S_2, the topological relationships "right of", "below", "above", and "left of" can be defined as follows (topological relationship is the relationships between the modules in the packing is independent of module sizes):

$$(a...c, a...c) \rightarrow a \text{ is left to } c$$
$$(a...c, c...a) \rightarrow a \text{ is above } c$$

Consequently, given a sequence pair (X, Y), the horizontal relationship among the blocks is known and can be represented by a horizontal-constraint graph $G_h(V, E)$. Similarly, the vertical relationship among the blocks can also be represented by a vertical-constraint graph $G_v(V, E)$. An example is shown in figure 2.8. Both graphs are vertex weighted, directed, and acyclic, so the longest path algorithm can be applied to determine the x and y coordinates of each block where the coordinates of a block are the coordinates of its lower left corner. As a result, the width and height of the chip are determined as the longest path length between the source and the sink in G_h and G_v, respectively [21].

Sequence Pair approach can be extended to represent a packing of arbitrarily shaped rectilinear blocks [15, 45]. Each arbitrarily shaped rectilinear block is decomposed into a set of L-shaped blocks. They show that the algorithm achieves placements with excellent area utilization [45]. Moreover, the average time and space complexity of SP can be reduced by using the Directed View Algorithm [19]. Thus, sequence pair becomes more attractive to be used in placement-related optimization problems.

Sequence Pair (SP) can handle all slicing and non-slicing structure and it is very flexible in representation. It has a good performance in handling modules with rectilinear shapes [45, 15]. However, the construction time and solution space are large. The solution space is large because when the relative position

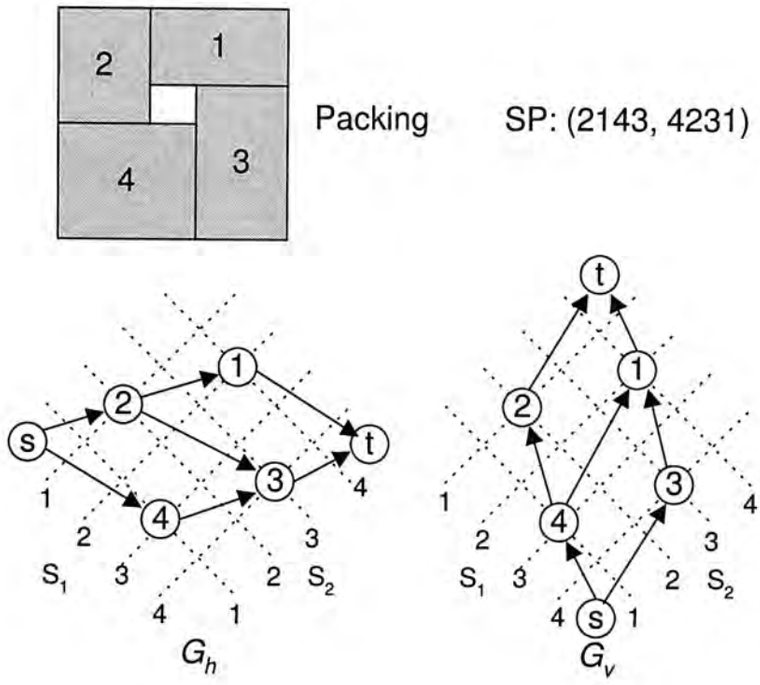


Figure 2.8: Horizontal and vertical constraint graph for a SP

of two blocks is both up and right, their relative position in the sequence pair can have multiple choices. This redundant representations will cause a one-to-many mapping from a floorplan to the representations. As a result, some extended versions of SP are devised that can reduce the time complexity of floorplan realization from $O(n^2)$ to $O(n^{1.25})$ [19] or $O(nlogn)$ [37, 18].

2.2.3 Bounded-Sliceline Grid (BSG) Structure

BSG (Bounded-Sliceline Grid) was first introduced for packing of rectangular modules in paper [23]. It consists of horizontal and vertical bounded segments, called the BSG segments. The segments dissect the plane into rooms associated with binary relations "right-to" and "above" such that any two rooms are related uniquely in either one of the two ways. Modules are assigned into the rooms, with at most one in one room. Then one dimensional horizontal and vertical compaction will be used to pack the modules closely together. An

example is shown in figure 2.9.

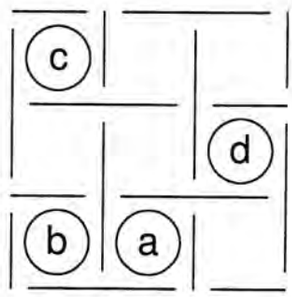


Figure 2.9: Structure of BSG

In general, two directed acyclic graphs, horizontal BSG-constraint graph $G_h(V_h, E_h)$ and vertical BSG-constraint graph $G_v(V_v, E_v)$ are constructed from each BSG. From the graphs, the positions of the modules can be determined easily. The length and width of the packing can also be computed similarly. An example is shown in figure 2.10.

The paper [24] shows that BSG based packing algorithm can handle pre-placed modules and rectilinear modules. In addition, a floorplanner that can handle convex-rectilinear blocks is developed also based on the BSG representation [32]. It can handle all slicing and non-slicing structure but it has the same problem as the SP that the construction time and the solution space are large and many redundancies exist.

2.2.4 O-Tree

An n-node O-tree is a tree with $n + 1$ node encoded by (T, π), where T is a $2n$-bit string that identifies the structure of the tree, and π is a permutation of the n node labels. When constructing T by traversing the tree [10], a '0' represents descending an edge and a '1' represents ascending an edge. An example is shown in figure 2.11.

The nodes of an O-tree represent rectangular blocks and the structure of

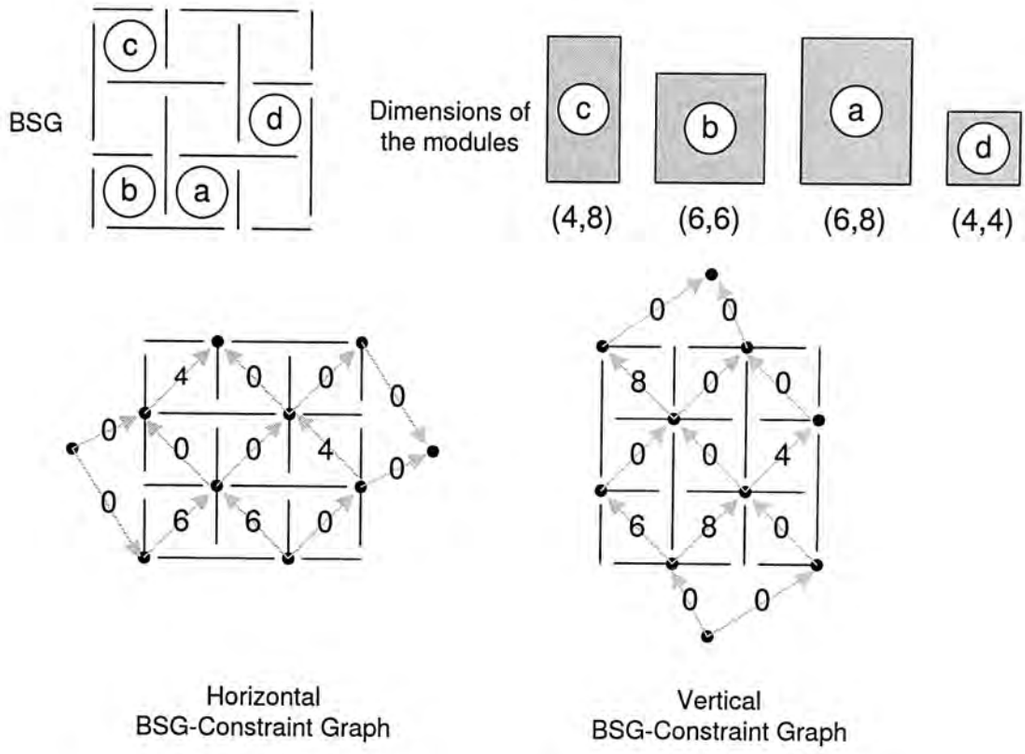
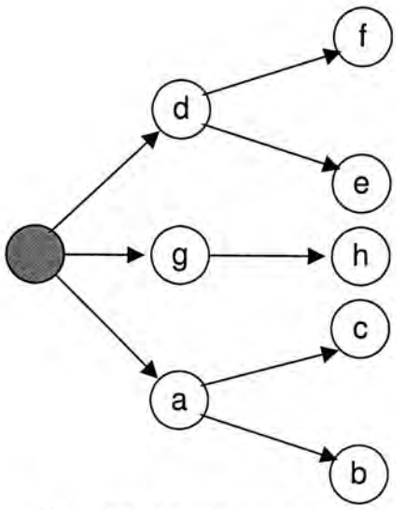


Figure 2.10: Constructing constraint graphs from a BSG

the tree imposes positioning constraints between the modules both horizontally and vertically [10]: Each parent block must be placed to the left of its children and if two blocks overlap in their x-coordinate projection, the block having a larger index in the permutation π must be placed on top of the one having a smaller index. An example is shown in figure 2.12.

O-Tree can handle non-slicing structure. It has a smaller solution space and construction time. As a result, it is a very effective representation for non-slicing floorplan. In addition, the O-tree based placement tool [30] has good performance when dealing with analog designs taken from industry. However, the tree structure is irregular so it is more difficult to implement than BSG or SP in representation. In addition, it is not a topological representation, i.e., the packing transformed from the representation is dependent on block sizes.



$T = 0010110011001011, \pi = abcghdef$

Figure 2.11: The O-Tree and its representation string

2.2.5 B*-Tree

A B*-tree [4] is an ordered binary tree (figure 2.13) whose root corresponds to the module on the bottom-left corner. The B*-tree can be constructed for an admissible placement P in a recursive fashion (admissible placement means that no blocks can shift left nor bottom when other blocks do not move). Starting from the root, the left sub-tree is constructed first and followed by the right sub-tree. Let R_i denote the set of modules located on the right-hand side and adjacent to b_i. The left child of the node n_i corresponds to the lowest module in R_i that is unvisited. The right child of n_i corresponds to the module located above and adjacent to b_i, with its x-coordinate equal to that of b_i and its y-coordinate less than that of the top boundary of the module on the left-hand side and adjacent to b_i.

If node n_j is the left child of node n_i, module b_j must be located on the right-hand side and adjacent to module b_i in the admissible placement, i.e., $x_j = x_i + w$ where w is the width of the module i. Besides, if node n_j is the right child of n_i, module b_j must be located above and adjacent to module b_i,

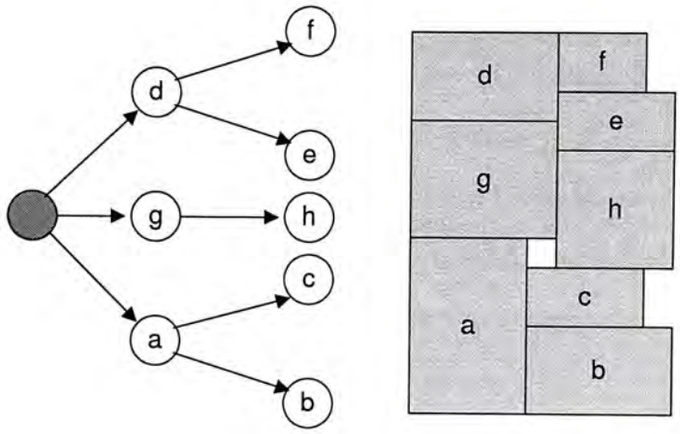


Figure 2.12: An O-tree and its corresponding block placement

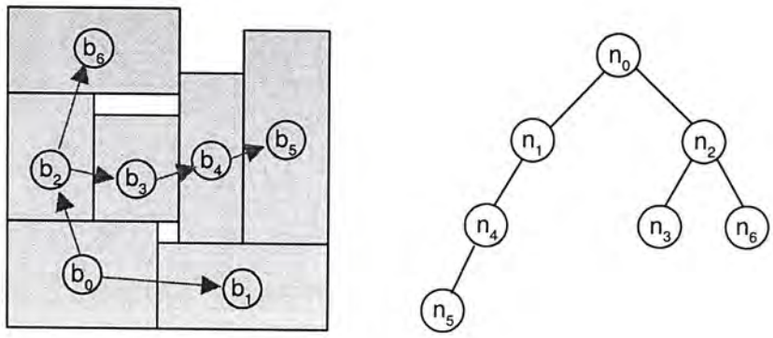


Figure 2.13: A B*-Tree and its corresponding block placement

with the x-coordinate of b_j equal to that of b_i, i.e., $x_j = x_i$. The coordinates of the root is (0,0), and the geometric information of any module can be obtained easily without constructing the constraint graphs for area cost evaluation.

B*-Tree can handle non-slicing structure. It also inherits many nice properties from binary trees and thus are very simple, efficient, and flexible for manipulating various types of modules and constraints directly and incrementally. It has a small solution space and construction time. As a result, it is a very effective representation for non-slicing floorplan. It is flexible to deal with hard, pre-placed, soft and rectilinear modules. However, like O-tree, B*-tree is not a topological representation.

2.2.6 Corner Block List

Corner block list (CB) [12] is constructed from a sequence of a recursive corner (upper-right) block deletion. For each block deletion, the block name, corner block orientation and number of T-junctions uncovered are kept. (S: list of block names; L: list of block orientations; T: number of T-junctions uncovered) After all the modules are deleted, the items in these three lists are concatenated in a reverse order. An example is shown in figure 2.14.

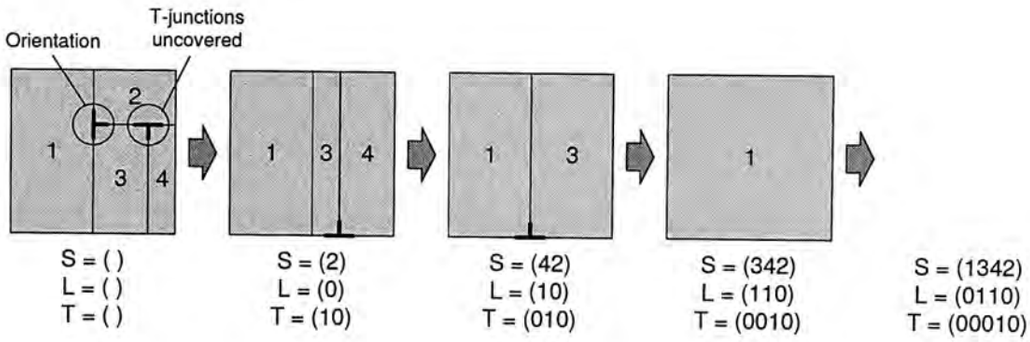


Figure 2.14: Example of transformation from floorplan to Corner Block List

Corner block list can handle mosaic floorplans and slicing floorplans. The time complexity of floorplan realization of corner block list is much lower than the other representations such as SP and BSG. In addition, the packing transformed from a corner block list is independent of the block sizes. As a result, it is an effective representation for mosaic floorplans and slicing floorplans. However, it cannot represent a floorplan that has two T-junctions meeting at the same point and some corner block lists do not correspond to any valid packing. For example, $S = (ABC), L = (111), T = (0100)$ is a valid corner block list but there is no floorplan corresponding to it.

Corner block list can be extended by including dummy blocks of zero area in the set of modules but it will use $O(n^2)$ dummy blocks and the size of the solution space will be increased significantly.

2.2.7 Twin Binary Tree

In the paper [46], Yao, et al. first suggest that Twin Binary Tree (TBT) can be used to represent mosaic floorplan. They have shown a one-to-one mapping between mosaic floorplan and TBT.

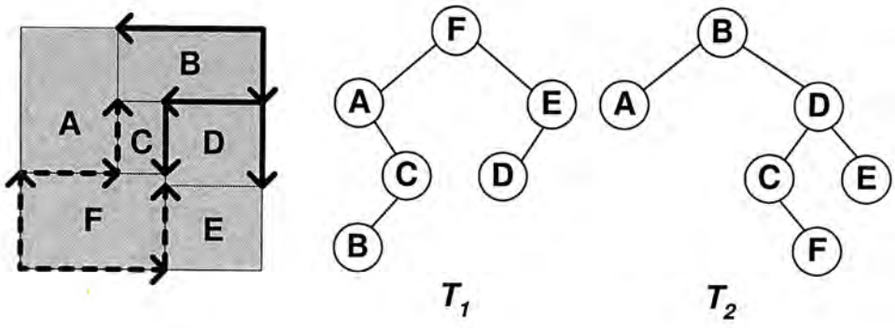


Figure 2.15: Building a pair of twin binary trees from a mosaic floorplan

Given a mosaic floorplan, we can obtain a pair of twin binary trees T_1 and T_2 by traveling along the slicelines of the floorplan. An example is shown in figure 2.15. To construct T_1, we can start from the module at the lower left corner and travel upward and to the right. Whenever the lower left corner of another module x is reached, a node labeled x is inserted into the tree and the process will be repeated by starting from the module x until all the modules in the floorplan are visited. Similarly, we can construct T_2 by starting from the module at the upper right corner and travel downward and to the left. Whenever the upper right corner of another module y is reached, a node labeled y is inserted into the tree and the process will be repeated by starting from the module y until all the modules in the floorplan are visited.

A floorplan can be constructed from a TBT efficiently in linear time and the solution space is much smaller. However, Twin Binary Tree can represent the mosaic floorplan only.

Later, the paper [47] proposed the Twin Binary Sequence (TBS) based on TBT. They have made use of mosaic floorplan as an intermediate step

to represent non-slicing structure. Instead of including an extra number of dummy blocks in the set of modules, TBS 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 and the construction time is still linear.

2.2.8 Comparisons between Different Representations

There are many different ways to represent a floorplan. In this section, a general comparison on their solution space is shown in table 2.1 and a comparison on flexibility and efficiency is shown in table 2.2

Representation	Solution Space	Slicing Floorplan	Mosaic Floorplan	General Floorplan
Polish Expressions	$O(\frac{n!2^{3n-3}}{n^{1.5}})$	Yes	No	No
Sequence Pair	$O((n!)^2)$	Yes	Yes	Yes
BSG	$O(n!C_n^{n^2}))$	Yes	Yes	Yes
O-Tree	$O(\frac{n!2^{2n-2}}{n^{1.5}})$	Yes	Yes	Yes
B*-Tree	$O(\frac{n!2^{2n-2}}{n^{1.5}})$	Yes	Yes	Yes
Corner Block List	$O(n!2^{3n-3})$	Yes	Yes	No
Twin Binary Sequence	$O(\frac{n!2^{3n-3}}{n^{1.5}})$	Yes	Yes	No

Table 2.1: Comparisons of different floorplan representations on solution space

2.3 Algorithms of Floorplan Design

There are many floorplanning algorithms using different methods. In this section, some algorithms are briefly explained and described.

Representation	Topological Representation	Memory Usage	Floorplan Realization Time
Polish Expressions	Yes	$O(n)$	$O(n)$
Sequence Pair	Yes	$O(n)$	$O(nlogn)$
BSG	Yes	$O(n^2)$	$O(n^2)$
O-Tree	No	$O(n)$	$O(n)$
B*-Tree	No	$O(n)$	$O(n)$
Corner Block List	Yes	$O(n)$	$O(n)$
Twin Binary Sequence	Yes	$O(n)$	$O(n)$

Table 2.2: Comparisons of different floorplan representations on flexibility and efficiency

2.3.1 Constraint Based Floorplanning

In Constraint based floorplanning [39], the floorplan is required to satisfy a set of horizontal and vertical topological constraints derived from an intermediate floorplan solution that may contain overlapping modules. In order to construct a floorplan with optimal area under the constraints, two directed acyclic graphs: horizontal constraint graph and vertical constraint graph will be constructed. The constraint graphs of a valid floorplan can be obtained by removing the conflicting constraining edges and the area of the final floorplan can be further optimized by reshaping of the flexible blocks.

The constraints will limit the floorplanner to explore a large number of solutions for area optimization but they can ensure that the final floorplan solution will inherit from the intermediate floorplan solution some important properties like small timing delay.

2.3.2 Integer Programming Based Floorplanning

In integer programming based floorplanner, the floorplanning problem is formulated as a set of linear equations using 0/1 integer variables [36]. Two types of constraints are considered: the non-overlapping constraints and the

routability constraints. The non-overlapping constraints prevent overlapping of the blocks and the routability constraints estimate the routing area required between the blocks and ensure all wires can be routed successfully by passing through the routing area. For critical nets, their net lengths are constrained to be within a bound. The length of a net depends on the timing budget of that net. These critical net constraints ensure that the lengths of the critical nets do not exceed the specified value.

2.3.3 Neural Learning Based Floorplanning

The floorplanning of mixed hard and flexible blocks is a complex combinatorial problem. Artificial neural networks (ANN) have been reported [49] as potential optimization tools in floorplanning. There are two main approaches: optimization by modeling as Hopfield networks and self-organization map.

2.3.4 Rectangular Dualization

Since the output from the partitioning stage can be represented as a graph $G = (V, E)$ where the vertices of the graph correspond to the sub-circuits and the edges represent the interconnections between the sub-circuits. A floorplan can be obtained by converting this graph into its rectangular dual. A rectangular dual of a graph $G = (V, E)$ consists of non-overlapping rectangles that satisfy the following properties: Each vertex v_i in V corresponds to a distinct rectangle R_i, $1 \leq i \leq |V|$, and for every edge (v_i, v_j) in E, the corresponding rectangles R_i and R_j are adjacent in the rectangular dual. The use of rectangular dualization [16] can maximize adjacency of blocks that are heavily connected or connected by critical nets.

2.3.5 Simulated Annealing

Simulated annealing is an iterative and non-deterministic technique. In simulated annealing, all moves that result in a decrease in cost are accepted. Moves that result in an increase in cost are accepted with a probability that decreases over the iterations. The detailed algorithm is shown in figure 2.16.

Simulated Annealing

$temp = init_temperture$
$place = init_placement$
while $temp > final_temperature$ do
  while $inner_loop_criterion$ = FALSE do
    new_place =PERTURB($place$)
    $\triangle C$ =COST(new_place)−COST($place$))
    if ($\triangle C < 0$) and RANDOM(0,1) $> e^{\frac{-\triangle C}{temp}}$
      $place = new_place$
    $temp$ =SCHEDULE($temp$)

Figure 2.16: Simulated Annealing

2.3.6 Genetic Algorithm

Genetic algorithm is also an iterative technique. It is an effective optimization and searching technique inspired by the mechanism of evolution and natural genetics. A parallel search is achieved by keeping a set of possible solutions to the optimization problem, called population. The individuals in the population are evaluated by a fitness measure. Two individuals are selected at the same time based on their fitnesses. The genetic operations (crossover and mutation) are applied on the selected parents to generate new possible solutions.

2.4 Summary

In this chapter, we have reviewed the basic idea of floorplan design. In floorplanning stage, we will try to optimize the area and interconnect cost of the layout by different techniques such as simulated annealing and genetic algorithm. In addition, we have reviewed the representations of floorplans. If a representation can handle all general non-slicing floorplan, all possible floorplan solutions can be searched. However, the larger solution space may result in slower runtime.

Chapter 3

Literature Review on Interconnect-Driven Floorplanning

3.1 Introduction

In floorplanning, the size of the layout is widely used in measuring the quality of the floorplan. However, due to the advances in VLSI technology, the sizes of the transistors have been decreasing and their switching speeds increasing. This has increased the importance of interconnect delays with respect to the overall speed performance of the circuit. As a result, floorplanning should also be a timing sensitive stage. In this chapter, we will review some interconnect-driven floorplanners.

3.2 Simulated Annealing Approach

3.2.1 "Pepper - A Timing Driven Early Floorplanner"

In this paper [25], the floorplanning system is designed to be highly flexible by allowing the users to specify various objectives (wirelength reduction, chip

area reduction, routability and timing) using a variety of manual floorplanning functions. It uses a simulated annealing based approach for the floorplanning algorithm because of the flexibility provided by simulated annealing.

In this floorplanner, a timing analyzer is incorporated into the floorplanner that can do timing analysis to provide feedback to the floorplanning algorithm. The timing analyzer is able to operate in an incremental fashion so that the analysis can be performed extremely fast.

The result shows that better clock speed can be obtained when both timing and wirelength are optimized simultaneously.

3.2.2 "A Timing Driven Block Placer Based on Sequence Pair Model"

In this paper [13], the floorplanning system uses SP as the floorplan representation and uses simulated annealing. The goal of this floorplanner is to optimize area and wirelength under the constraints in timing delay. The problem can be summarized as follows,

$$\begin{aligned} \text{Minimize:} \quad & Area(s) + \alpha \times Wirelength(s) \\ \text{Subject to:} \quad & Delay(s) \leq D \text{ and} \\ & R(s) = 0, \end{aligned}$$

where s is the configuration of a placement, α is the weight, D is the upper bound on the timing delay, $R(s) = max\{|Ratio - width(s)/height(s)|, |Ratio - height(s)/width(s)|\}$, and $Ratio$ is a constant to denote the desired aspect ratio. According the above constraints, the cost function is computed as follows,

$$Cost(s) = Area(s) + w1 * wirelength(s) + w2 * max(0, delay(s) - D) + w3 * R(s)$$

Instead of using half-perimeter in wirelength estimation, it selects the maximum Manhattan distance from the source to any other pins in the wirelength

estimation. Results show that there may be some area penalty in this floorplanner but all timing constraints can be satisfied.

3.2.3 "Integrated Floorplanning and Interconnect Planning"

In this paper [5], their floorplanner uses Polish expressions to represent floorplans and searches for an optimal floorplan solution using simulated annealing. To achieve a more accurate interconnect cost, it performs careful interconnect planning on the candidate floorplan solutions, instead of using the traditional wirelength estimation method. When the temperature is high, it uses the conventional half-perimeter routing estimation. When the temperature is medium, they use a more accurate routing estimation based on L-shaped routing. When the temperature is low, they use Z-shaped routing. Such approach can reduce the run time when the multi-stage simulated annealing approach is used. In that paper, the cost function used can be represented as follows,

$$COST = \alpha A + \beta W + \gamma OF,$$

where α, β, γ are constants, A is the total area of the packing, W is the half-perimeter estimation of the interconnect cost and OF is the number of squares with overflow in routings. The term OF is different in different stages because different routing estimation is used. Results show that the floorplanner has good performance in improving the routability.

3.2.4 "Interconnect Driven Floorplanning with Fast Global Wiring Planning and Optimization"

In this paper [3], the floorplanner is based on simulated annealing and uses Polish expression to represent a slicing floorplan structure. Besides considering

traditional measurements (total wirelength and chip area) for a good floorplanning candidate, it focuses on interconnect optimization and estimation with consideration of layer assignment, buffer insertion and wire sizing/spacing. Moreover, it uses multi-stage simulated annealing to reduce run time.

Global Wire Planning Under Area Constraint (GWP-A) is introduced in this paper. It can estimate the overall performance, the wire width and optimized spacing for each layer and the layer assignment.

In stage one, it uses a weighted cost function of area, total wirelength and maximum wirelength. It uses half-perimeter estimation to compute the net length. The maximum wirelength is used as a rough measure of the maximum interconnect delay. In stage two, the result computed by the GWP-A is used in the cost function instead of using the maximum wirelength. In stage three, the congestion information (Z-shaped routing) is added into the function. In the final stage, it uses global routing to obtain an even more accurate estimation of the congestion information. The result shows that this floorplanner can improve delay significantly with only a little area penalty.

3.3 Genetic Algorithm Approach

3.3.1 "Timing Influenced General-cell Genetic Floorplanning"

In this paper [31], a timing-driven genetic floorplanner is implemented and studied. It is divided into two phases. In the first phase, the solutions are restricted to slicing structure and the blocks are rigid but have free orientations. In the second phase, the blocks are flexible and the solutions can be non-slicing. The modules can maintain their topological proximity obtained from the result of the first phase.

In the first phase, Polish expression is used. After initializing the population, individuals are selected as parents for crossover and mutation. The selection mechanism is based on the proportionate selection scheme implemented by the roulette wheel method. In this selection scheme, each string is allocated a sector of a roulette wheel with the angle subtended at the center equaling to $2\pi \times \frac{f_i}{sumf}$ where f_i is the fitness of solution i.

$$Fitness(i) : f_i = \frac{A_i^*}{A_{max}} \times W_A + \frac{T_i^*}{T_{max}} \times W_T + \frac{L_i^*}{L_{max}} \times W_L,$$

where A^*, T^* and L^* represent the quantities of area, clock speed and wirelength respectively, W_A, W_T and W_L are user specific weights for area, timing and wirelength respectively, A_{max}, T_{max} and L_{max} are the maximum area, clock speed and wirelength in the given population respectively.

Crossover can inherit useful information from two individuals (parents) to offspring and mutation can introduce new information into the population. Finally, it selects the best individual and the remaining individuals may be selected based on their fitness (higher fitness translates to higher survival probability).

In the second phase, two graphs, horizontal constraint graph and vertical constraint graph are constructed. They are used in the reshaping algorithm to determine the dimensions and positions of the blocks so that the floorplan area is minimized.

3.4 Force Directed Approach

3.4.1 "Timing Influenced Force Directed Floorplanning"

In this paper [48], it obtains a floorplan in two steps: Construction of a timing driven topological arrangement using a force-directed approach and conversion of the topological arrangement into a legal floorplan. In the first step, the

topological arrangement is obtained in a greedy fashion by adding in a block at a time to the partial floorplan (point placement). In the greedy approach, it should select the block for placement based on a criteria of timing and connectivity. The block with maximum gain in timing and connectivity is selected for positioning. The positioning procedure considers the connections of the target block to the IO pads to compute its location.

Finally, two constraint graphs are constructed for area optimization by reshaping. In the reshaping, the algorithm determines dimensions and positions of the blocks so that the floorplan area is minimized and the constraints on block shapes are satisfied. Results show that this approach can increase the percentage of critical paths that can satisfy the timing constraint.

3.5 Congestion Planning

Congestion planning is also an important issue of floorplanning. The excessive congestion will result in a local shortage of routing resource. It means that there will be a large expansion in area, or even an unroutable design failing in achieving timing closure after detailed routing. Thus, a good estimation algorithm is required for congestion analysis during earlier stages of the design.

3.5.1 "On the Behavior of Congestion Minimization During Placement"

In this paper [40], a realistic global router is used to evaluate congestion of each candidate placement solution. It is because they find that minimization of wirelength is not equivalent to minimization of congestion.

3.5.2 "Congestion Minimization During Placement"

In both papers [41, 42], they propose a post processing stage with a very effective net-centric algorithm to reduce congestion in a layout. They try to minimize the wirelength first by using simulated annealing and then apply a net-centric algorithm to reduce the congestion of the layout. According to the algorithm, all nets are routed first and a weight is assigned to each net. The weight of a net is equal to the number of overflow global edges that the net crosses. Then the nets are sorted in descending order according to their weights. In order to reduce the congestion, they will move a block that is connected to the net with the greatest weight to a new position which can result in a reduction in congestion at the most congested area. Results show that congestion can be reduced efficiently by this algorithm.

3.5.3 "Estimating Routing Congestion Using Probabilistic Analysis"

In this paper [20], a stochastic congestion estimation algorithm for a placed netlist that is blockage aware is presented. It first divides the floorplan into a grid structure and then calculate the probabilities of the two-pin wires passing through each grid for both the horizontal tracks and the vertical tracks. Assuming that a wire k is passing through from $(1,1)$ to (m,n), the number of possible routes passing through grid (i,j) along the horizontal track $T_x(i,j)$ and the vertical track $T_y(i,j)$ can be calculated as below:

$$T_x(i,j) = \begin{cases} F(m,n-1) & i=1, j=1 \\ 1 & i=1, j=n \\ F(m-i+1,n-1) & 1<i<m, j=1 \\ \frac{F(m,n-j+1)+F(m,n-j)}{2} & i=1, 1<j<n \\ \frac{F(i,j)F(m-i+1,n-j)+F(i,j-1)F(m-i+1,n-j+1)}{2} & \text{others} \end{cases}$$

$$T_y(i,j) = \begin{cases} F(m-1,n) & i=1, j=1 \\ 1 & i=1, j=n \\ \frac{F(m-i+1,n)+F(m-i,n)}{2} & 1<i<m, j=1 \\ F(m-1,n-j+1) & i=1, 1<j<n \\ \frac{F(i,j)F(m-i,n-j+1)+F(i-1,j)F(m-i+1,n-j+1)}{2} & \text{others} \end{cases}$$

where $F(m,n)$ is the total number of possible routes from $(0,0)$ to (m,n).

The authors show that the probabilistic congestion model correlates well to post-route results. Therefore, it can be an ideal candidate for the inner loop of synthesis and placement optimizations. However, buffer planning has not been considered in their algorithm.

3.6 Buffer Planning

In some previous studies [17, 26], it has been shown that buffer insertion is the most effective method to optimize signal delay and noise. However, buffer insertion should be planned during the process of floorplanning.

3.6.1 "Buffer Block Planning for Interconnect Driven Floorplanning"

In this paper [7], it introduces the concept of feasible region (FR) for buffer insertion. It was found that the FR of a buffer is very large and this provides a lot of flexibility to plan a buffer's location. Moreover, an effective buffer block-planning (BBP) algorithm is proposed and it is shown in figure 3.1.

Results show that this algorithm leads to a significant improvement in meeting delay constraints with small area penalty.

Buffer Block Planning

Build horizontal and vertical polar graph
Build tile data structure
For each tile, compute its area slacks
While (there exists buffer to be inserted)
  tile = **Pick_A_Tile()**
  Insert_buffers(*tile*)
Update width and height of chip, FR and area slacks

Figure 3.1: Algorithm of buffer block planning

3.6.2 "Routability Driven Repeater Block Planning for Interconnect-centric Floorplanning"

The paper [33] introduces the concept of independent feasible regions for buffers and derives an analytical formula to compute the positions of these regions. It develops a routability-driven buffer clustering algorithm to perform buffer block planning. In its congestion model, it employs a two dimensional rectangular grid based probabilistic map assuming two-bend routing for each segment. Assuming that the source is located at $tile(0,0)$ and the sink is located at $tile(m,n)$. The probability matrix can be derived as in table 3.1.

Tile	Number of horizontal routes passing through $tile(i,j)$	Number of vertical routes passing through $tile(i,j)$
$0 < i < m; 0 < j < n$	$\frac{1}{n+m+2}$	$\frac{1}{n+m+2}$
$0 < i \leq m; j = 0$	$\frac{n-i}{n+m+2}$	$\frac{1}{n+m+2}$
$i = 0; 0 < j \leq n$	$\frac{1}{n+m+2}$	$\frac{m-j}{n+m+2}$
$i = 0; j = 0$	1	1
$i = m; j = n$	1	1

Table 3.1: Probability Matrix

Finally, it uses a monotonic piecewise cost function as shown in figure 3.2 after all routing tiles have been assigned a congestion cost.

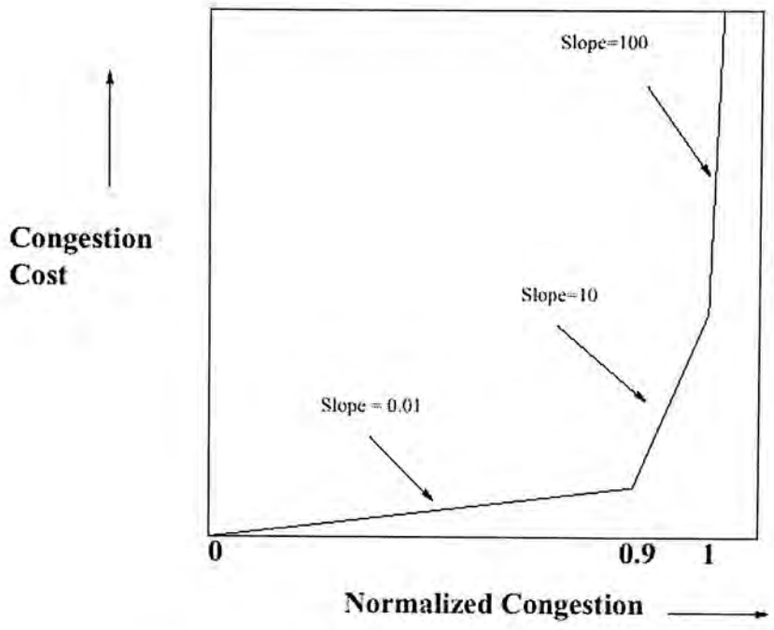


Figure 3.2: Piecewise-linear congestion cost function

3.6.3 "Provably Good Global Buffering Using an Available Block Plan"

In this paper [9], the authors address the problem of inserting buffers into global nets given an existing buffer plan. They assume that the global nets have been already decomposed into two-pin wires. Their method routes wires using available buffer blocks in such a way that buffers will be inserted at the intervals satisfying the upper and lower bounds requirements. In their method, more than one buffer is allowed to be inserted into any given wire. In addition, their algorithm observes buffer parity constraint that it will choose to use buffer according to source and destination signal parity.

3.6.4 "Planning Buffer Locations by Network Flows"

In the paper [38], Tang and Wong propose an optimal algorithm to assign buffers to buffer rooms by network flow [1]. First, they compute the buffer

rooms which are disjoint regions bounded by the boundaries of the feasible regions defined in [7] and the buffer zones which are the partitions of the free space. They can then construct a network flow graph and assign the capacities to the edges according to the area of intersection between the buffer rooms and buffer zones and the area of buffer zones. Finally, they can obtain the corresponding buffer planning solution by running the min-cut max-flow algorithm. As the independent feasible region is introduced in [33], this optimal algorithm for buffer block planning can be applied to simultaneously place all buffer without the need of considering only one buffer per net.

3.6.5 "A Practical Methodology for Early Buffer and Wire Resource Allocation"

In the paper [2], Alpert et al. make use of tile graph and propose the four-stage RABID (Resource Allocation for Buffer and Interconnect Distribution) heuristic to perform buffer block planning by using dynamic programming. First, a steiner tree is constructed for each net in order to give an initial routing of each net. Then, the floorplanner will rip-up-and-reroute the initial steiner trees to reduce the wire congestion. The floorplanner allocate buffers to each net in the order of net delay. During the buffer assignment process, the probability of successful assignment at each tile which is along the route of the net is computed first. The cost of different combinations of buffer insertions can then be computed according to the probabilities. Finally, the combination of buffer insertions with minimum cost will be chosen and the buffers will be assigned to the tiles accordingly.

3.7 Summary

In floorplanning, due to the advances in VLSI technology, the sizes of the transistors have been decreasing and their switching speeds increasing. This has increased the importance of interconnect delays with respect to the overall speed performance of the circuit. As a result, floorplanning should also be a timing sensitive stage. In the papers [31, 13, 48, 5, 3], they have formulated different congestion-related cost functions (always evaluated by some simple global routing) including a hybrid length plus congestion cost function and optimize these cost functions by applying heuristics such as simulated annealing and genetic algorithm.

However, the paper [42] have stated that using the congestion-related cost function cannot reduce congestion effectively. It is because simple global routing may have a large difference with the final detail routing. Thus, they proposed a post processing stage to minimize congestion by applying a net-centric algorithm. Besides that, the paper [20] presents a stochastic congestion estimation algorithm for a placed netlist that is blockage aware. The congestion is estimated by probabilistic analysis instead of by performing global routing directing. The algorithm is fast but buffer insertion is not considered.

Besides, buffers contribute delay and area, so their locations should be planned carefully. Consequently, the paper [7] uses buffer block planning algorithm to achieve timing improvement with small area penalty. Then the paper [33] uses a better heuristic in buffer block planning to achieve a better timing improvement. The area penalty can be further reduced by deleting the redundant buffers in the final stage. However, it requires a long run time since the new buffer blocking planning algorithm is complicated. In addition, the paper [38] proposed that buffers should be allowed to be inserted inside macro blocks and their method will distribute buffer sites all over the layout.

Chapter 4

Floorplanner with Fixed Buffer Planning [34]

4.1 Introduction

Traditional floorplanners have not paid enough attention to interconnect optimization. This results in a large expansion in area, or even an unroutable design failing in achieving timing closure after detailed routing. Buffer insertion is one of the most popular and effective techniques to achieve timing closure. In the current practice, buffers are inserted after routing. However buffers also take up silicon resources and cannot be inserted wherever we want. A good planning of the module positions during the floorplanning stage so that buffers can be inserted wherever needed in the later routing stages will be useful. Besides, buffer itself contributes delay and area, and their locations should be carefully planned.

In our method, we will compute the buffer usage of each net and estimate congestion taking into account buffer locations and net routability. We employ a simple yet accurate method to estimate congestion from a more global and realistic perspective. The nets are assumed to be routed in multi-bend shortest Manhattan distance and issues like buffer locations and blocked nets (a net that cannot be routed in the shortest Manhattan distance) will also be considered

in the evaluation of the floorplan. In order to consider the buffer issues in congestion estimation, we will compute the exact buffer locations for each net by a table-lookup approach.

4.2 Overview of the Floorplanner

In this section, we will present the routability-driven floorplanner with fixed buffer block planning in which fixed buffer insertion constraint [34] is employed. We assume that the wires are routed over-the-cell and multi-bend routing is used. We divide a floorplan into grids and the deadspace in each grid is calculated for estimation of the amount of buffer resources. We use sequence pair representation to describe a non-slicing floorplan solution.

Unlike traditional floorplanners, we will consider buffer usage of each net and estimate the wiring congestion of each intermediate solution taken into account the buffer requirements. In each iteration of the annealing process, we will compute the buffer requirements of each net and then estimate the wiring congestion of the floorplan solution considering both the module positions and the possible buffer locations. A floorplan solution will be evaluated according to the total chip area, total wirelength, congestion and availability of buffers.

4.3 Congestion Model

In this congestion model, we will estimate the congestion and routability of each intermediate packing solution. We assume that there are only two physical layers: one for the interconnection wires and the other one for the modules and buffers. In order to improve the routability of the design, our approach will try to reduce the congestion of the interconnect at different locations of the layout taken into accounts buffer locations and routing blockages.

4.3.1 Construction of Grid Structure

We want to obtain the congestion information at every location of a floorplan. In order to achieve this purpose, we simply divide the floorplan into a 2-dimensional array of grids (figure 4.1). Then we will obtain the congestion information at each grid.

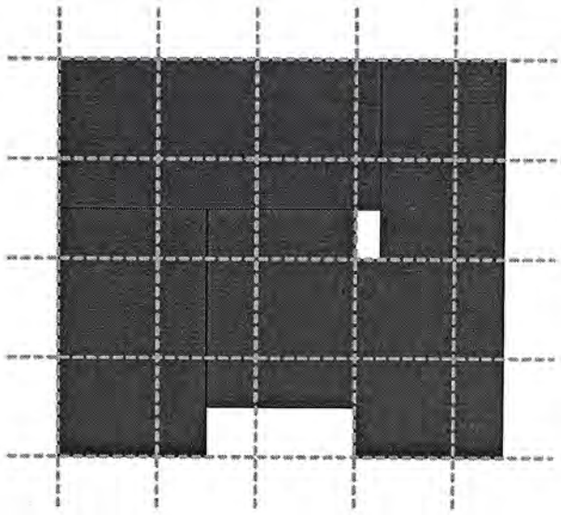


Figure 4.1: Division of floorplan

The number of grids is an important factor in our floorplanner. If the number of grids is small (e.g., if number of grids is equal to one, it means that there is no division), the area of each grid is large and one grid may cover too many modules. The congestion measured at each grid may become less informative. However, if the number of grids is too large, the speed of the whole implementation will be reduced greatly. In our implementation, 20x20 grids have been chosen.

There are two kinds of grids: blocked and unblocked. If a grid is covered by a module, no buffer will be allowed to be inserted into that grid and the grid will thus be marked as blocked (figure 4.2). We can mark all the blocked grids easily by scanning the sizes and the locations of all the modules once.

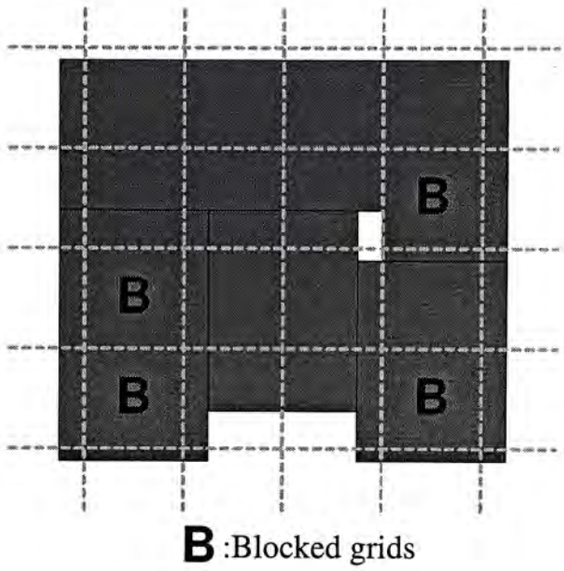


Figure 4.2: Examples of blocked grids

4.3.2 Counting the Number of Routes at a Grid

After dividing a floorplan into grids, we will count the number of routes of a net passing each grid. In our floorplanner, we assume multi-bend routing with shortest Manhattan distance. An example is illustrated in figure 4.3. In this example, we assume that the starting grid is on the upper-left side of the ending grid. At the beginning (figure 4.3(a)), the number of possible routes at each grid lying vertically or horizontally with the starting grid is initialized to one. Then, the values of the other grids can be computed dynamically row by row by adding up the values on its top and on its left. In general, the value $g_1(x, y, k)$ computed at the grid (x, y) is the number of possible routes for net k that go from the starting grid to the ending grid and pass through the grid at position (x, y). We can then obtain the total number of possible routes from the starting grid to the ending grid by filling in the values at all the grids. In order to compute the number of routes passing through each grid, we need to repeat the same steps from the ending grid to the starting grid (figure 4.3(d)) and compute $g_2(x, y, k)$, which is the number of possible routes for net

k going from the ending grid to the starting grid and passing through the grid at position (x, y). Then the number of routes passing through the grids at (x, y) will be computed as:

$$\text{Number of possible routes for net } k \text{ passing through the grid at } (x, y)$$
$$= g_1(x, y, k) \times g_2(x, y, k)$$

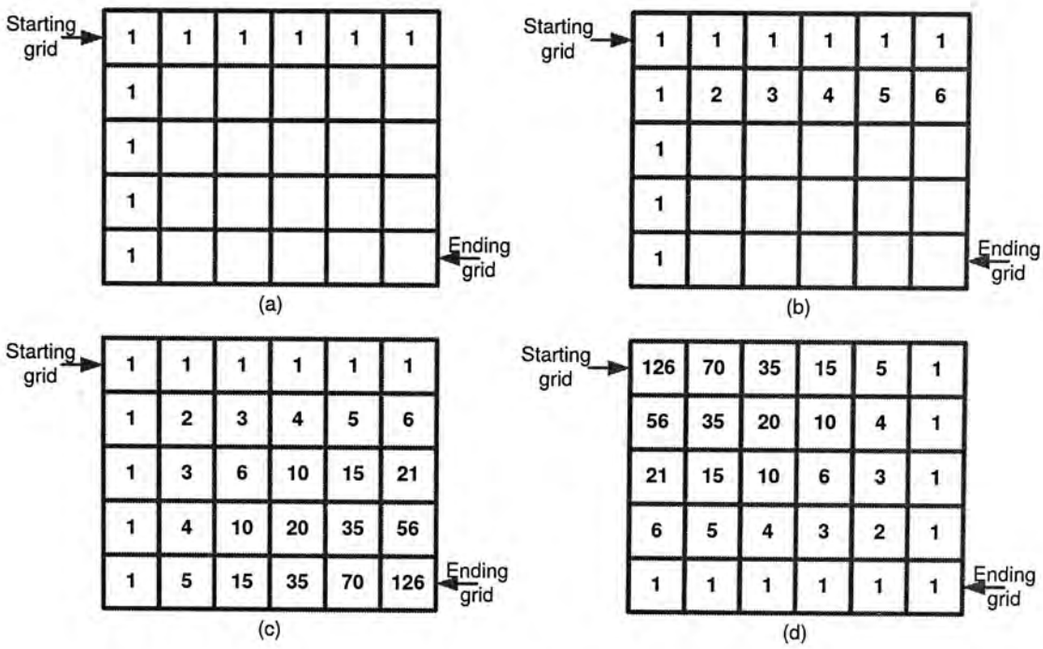


Figure 4.3: Example for counting the number of routes for a net

4.3.3 Buffer Location Computation

In our floorplanner, we will use Elmore Delay Model to compute the buffer locations. Elmore Delay Model is commonly used for delay estimation. For a wire divided into segments and with a load capacitance C_L and a load resistance R_L. If buffers are inserted in each segment, its delay can be computed by the following formula:

$$D = \sum_{k=1}^{n} [R_{B_{k-1}}(c_0 l + c_f l + C_{B_k}) + r_0 l(\frac{c_0 l}{2} + \frac{c_f l}{2} + C_{B_k}) + d_{B_k}] \quad (4.1)$$

where c_0, r_0 and c_f are unit wire capacitance, resistance and fringing capacitance respectively; $R_{B_0} = R_L$, $C_{B_n} = C_L$ and $d_{B_n} = 0$; l is the length of a wire segment, R_B, C_B and d_B are resistance, capacitance and intrinsic delay of a buffer respectively. Since the floorplan solution is divided into grid structures for congestion estimation, we will use the grid unit length as the length of a wire segment in the computation of buffer locations. The configuration of the parameters used in experiments are shown in following table 4.1.

$R_0(\Omega/mm)$	0.075	$R_B(\Omega)$	180
$C_0(fF/mm)$	0.118	$C_B(fF)$	23.4
$C_f(fF/mm)$	0.064	$D_B(ns)$	36400

Table 4.1: Configuration of experiment

4.3.4 Counting Routes with Blocked Grids

In this floorplanner, we assume that the information about the locations of the buffers for a net is already known. However, the wire must not pass through a blocked grid if a buffer should be placed there. An example is shown in figure 4.4.

When a net k passes through a blocked grid (x, y) at a position where a buffer should be placed, the possible number of routes at such a grid (x, y) (both $g_1(x, y, k)$ and $g_2(x, y, k)$) must be zero. An example is illustrated in figure 4.5.

Comparing figure 4.3 and figure 4.5, the total number of possible routes is reduced from 126 to 48 because of the lacking of routing resources at the blocked grids. It may happen that the number of possible route is equal to zero. It means that the wire cannot be routed from the starting grid to the ending grid with the shortest Manhattan distance, and we call such a wire a blocked wire.

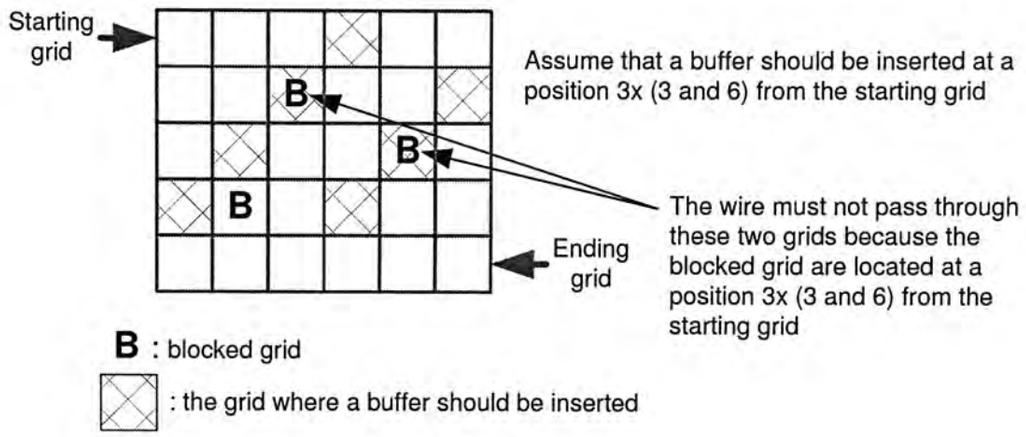


Figure 4.4: Example of blocked grid overlaps with the location of buffer insertion

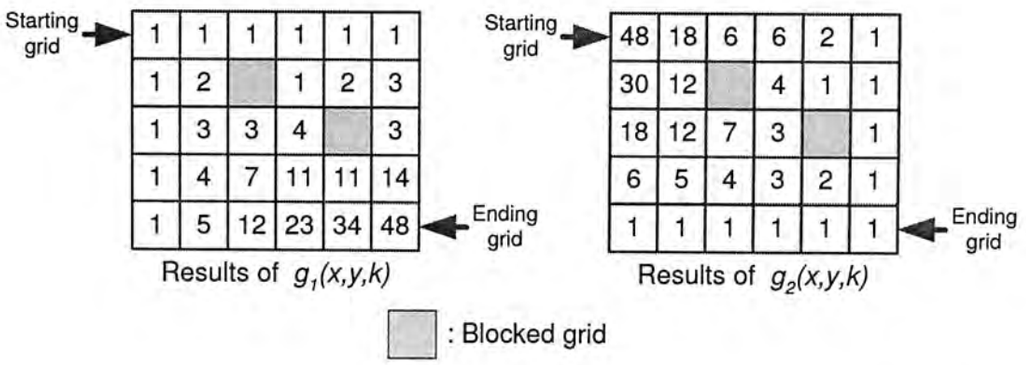


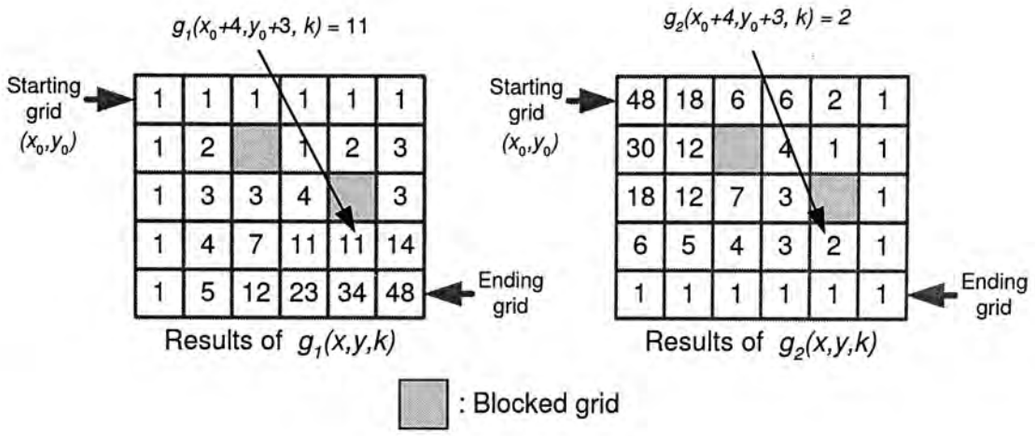
Figure 4.5: Results of $g_1(x, y, k)$ and $g_2(x, y, k)$ with blocked grids

4.3.5 Computing the Probability of Net Crossing

After counting the number of possible routes for a net, we can calculate the probability that the wire of a net will pass through any particular grid. The probability that a net k will pass through a grid at (x, y) is:

$$P(x, y, k) = \frac{g_1(x,y,k) \times g_2(x,y,k)}{g_2(x_0,y_0,k)}$$

where $g_1(x_0, y_0, k)$ is the total number of possible route from the starting grid to the ending grid. An example is shown below in figure 4.6.



$$
\begin{aligned}
& P(x_0+4, y_0+3, k) \\
= & \frac{g_1(x_0+4, y_0+3, k) \times g_2(x_0+4, y_0+3, k)}{g_2(x_0, y_0, k)} \\
= & \frac{11 \times 2}{48} \\
= & \ 0.458333
\end{aligned}
$$

Figure 4.6: Example of calculating $P(x, y, k)$ for net k at grid (x, y)

Finally, the average number of wire passing through a grid at (x, y) is computed as:

$$weight(x, y) = \sum_{netk} P(x, y, k)$$

If the value of $weight(x, y)$ is large, it means that the grid at (x, y) is very congested.

4.4 Time Complexity

According to this congestion estimation model, we need to scan the array at each grid from the source to the destination for each net. Therefore, the time complexity is $O(m \times k \times up)$ where k is number of grids scanned for each net, m is number of nets and up is the upper bound of the buffer insertion constraint.

If the floorplan is divided into $K \times K$ grids, we need to scan at most K^2 grids for each net. Moreover the upper bound in the buffer insertion constraint is also bounded by the length of the wires. (It is bounded by $2K$). Therefore, the time complexity of this congestion estimation is only $O(m \times K^3)$.

4.5 Simulated Annealing

Simulated annealing is an iterative and non-deterministic optimization technique. In simulated annealing, all the moves that result in a decrease in cost will be accepted. The moves that result in an increase in cost are accepted with a probability that decreases over the iterations. The algorithm is shown on figure 4.7.

Simulated Annealing

temp = *init_temperture*
place = *init_placement*
while *temp* > *final_temperature* do
  while *inner_loop_criterion* = FALSE do
    new_place =PERTURB(*place*)
    $\triangle C$ =COST(*new_place*)−COST(*place*))
    if ($\triangle C < 0$) and RANDOM(0,1) > $e^{\frac{-\triangle C}{temp}}$
      place = *new_place*
    temp =SCHEDULE(*temp*)

Figure 4.7: Simulated Annealing

In this floorplanner, the simulated annealing process is used and the cost function is shown below:

$$Cost = Area + \alpha \times Wire + \beta \times M_weight + \gamma \times B_wire$$

where *Area* is the area of the floorplan, *Wire* is the total wirelength, (it will be discussed in section 4.6) *M_weight* is the average weight of the top ten

percent most congested grids and B_wire is the number of blocked wires, α, β and γ are parameters. These parameters will be set at the beginning of simulated annealing process in such a way that the ratio of importance of the area term, the wirelength term, the congestion term and the blocked wire term will remain a constant. This can be done by performing a sequence of random walks at the beginning of simulated annealing process and sampling the average values of these penalty terms. The parameters α, β and γ are then computed accordingly.

4.6 Wirelength Estimation

In the floorplanning stage, the positions of the I/O pins in the modules are not yet fixed and the modules will usually cover a number of grids. Therefore, the assumption on the locations of the I/O pins will affect the results significantly in our method. Consequently, we need to estimate reasonably the positions of the I/O pins in order to compute the interconnect costs and congestion cost accurately.

In traditional floorplanner, half-perimeter estimation using the center of a module as the location of that module is most commonly used. However, if we consider congestion and buffer insertion in our floorplanner, center-to-center estimation is difficult to apply. It is because if a module M is large, center-to-center estimation will increase the congestion estimation at the center of M significantly. In this case, the congestion may be over estimated and this will reduce the accuracy of the whole method. In addition, buffer may be required to be inserted at the grids covered by module M. Therefore, many routes will become impossible as buffers cannot be inserted on macro blocks. As a result, we adopt a more appropriate method for estimating the positions of the I/O pins on a module that can be used in our routability-driven floorplanners. We investigated three different solutions for solving this problem.

4.6.1 Center-to-center Estimation

The first method is to use center-to-center estimation. It is the simplest method for placing the I/O pins (figure 4.8). First, we need to calculate the center of the modules. Then the I/O pins are assumed to be placed at the grid where the center point is. In the real situation, the I/O pins will not be placed at the center of the module. The I/O pins should be placed on the edges of the modules. Moreover, this estimation method will make the center grid very congest although it may not be the case in the actual situation.

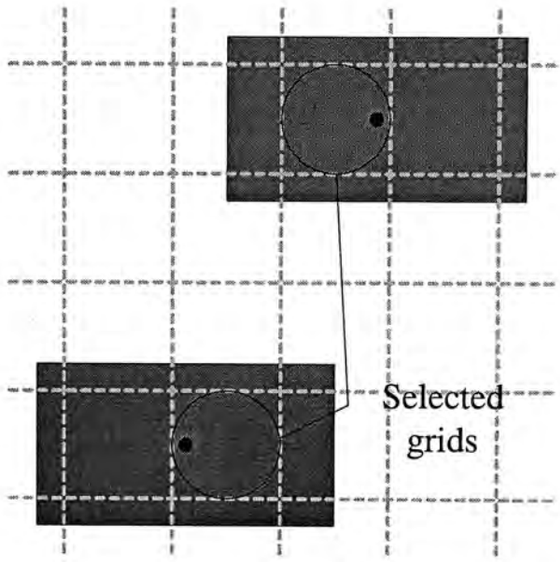


Figure 4.8: Center-to-center estimation

4.6.2 Corner-to-corner Estimation

The second method is corner-to-corner. From the center of the modules, four regions upper-left, upper-right, lower-left and lower-right are created. If one module (denote by A) is placed at the upper-right of another one (denote by B) such in figure 4.9.

The I/O pins connecting A and B will be placed at the grids where the

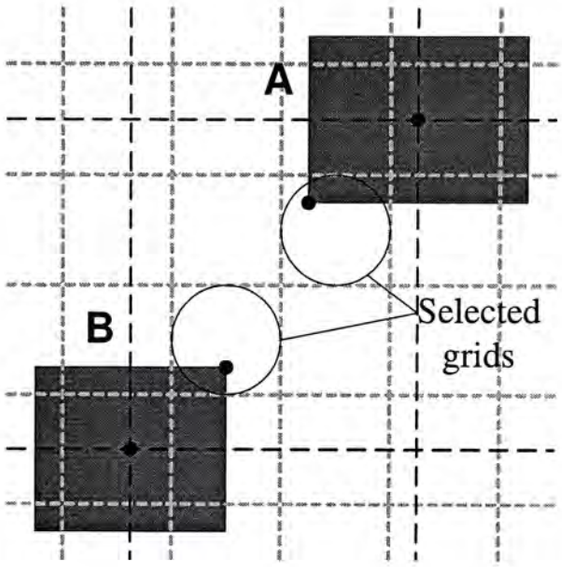


Figure 4.9: Corner-to-corner estimation

lower-left corner of A and the upper-right corner of B lie respectively. Corner-to-corner method is more accurate than the center-to-center method because all the I/O pins may be placed at the four grids instead of one and the I/O pins are placed on the edges of the modules.

4.6.3 Intersection-to-intersection Estimation

Finally, in order to distribute the I/O pins into the grids appropriately, intersection-to-intersection method is used. Consider a net connecting two modules A and B, we will first draw a line from the center of one module to another. The two intersecting points will be found on an edge of each module (figure 4.10) and the I/O pins will be placed at the grids containing the intersection points.

This is the most appropriate method for our floorplanner since the estimated positions of the I/O pins will be similar to those in the final design and the buffer locations can be estimated more accurately. However, extra calculations are needed which may lead to slower speed.

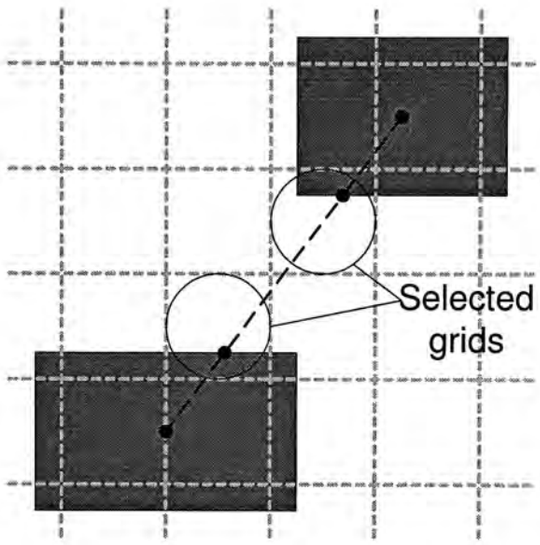


Figure 4.10: Intersection-to-intersection estimation

4.7 Multi-pin Nets Handling

In our design, buffer insertions can be applied when all the nets are two-pin wires only. For multi-pin nets, half-perimeter estimation cannot be used. In order to handle multi-pin nets, we need to decompose the multi-pin nets into a set of two-pin wires. There are several methods to decompose a multi-pin net into two-pin wires such as using minimum spanning tree (MST), rectilinear steiner tree (RST). MST runs faster but it may over-estimate the congestion because of the overlapping net segments. However, this conservative estimation will not affect the resultant packing significantly. Since the runtime of an RST algorithm is usually much slower than that of an MST algorithm, MST is a better choice for estimation purposes in the early floorplanning stage. As a result, we apply MST to handle multi-pin nets in our floorplanner.

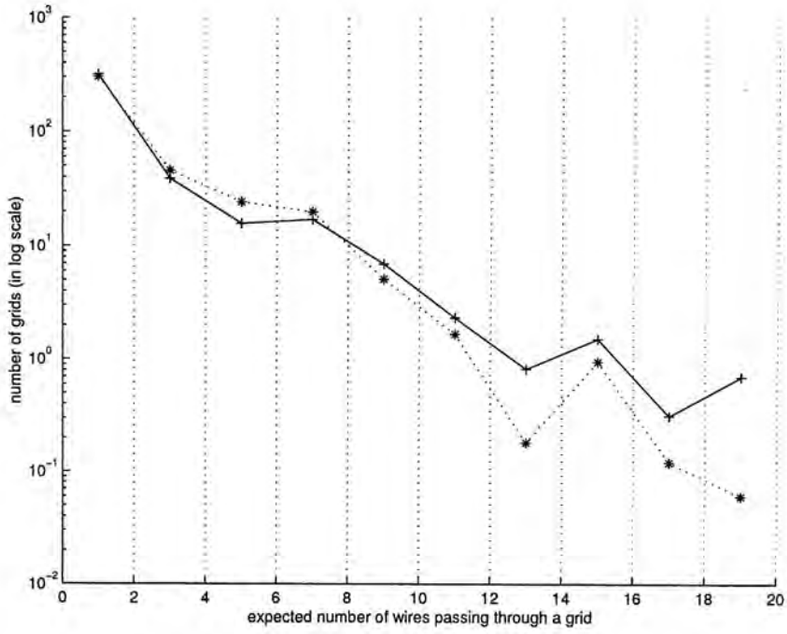


Figure 4.11: Distribution of wires among the grids for ami33

4.8 Experimental Results

We tested the floorplanners (f1: floorplanner with congestion estimation and f2: floorplanner without congestion estimation) using three MCNC building block benchmarks (*ami*33, *ami*49 and *playout*). The number of modules and nets are (33, 123), (49, 408) and (62, 1161) respectively. All the experiments were done using an UltraSPARC-II 400MHz processor with 8 GB memory. The packings will be divided into 20×20 grids and we assume that buffers are required at grids of distance 10x from the source of each net. Every set of results displayed in the tables is obtained by repeating the experiment eight times and picking the average from the eight trials.

In our experiments, the figure 4.11, 4.12 and 4.13 show the distribution of the wires among the grids in the three test cases. According to the graphs (dash line: f1; solid line: f2), we can see that the number of very congested grids for f1 is less than that for f2 and the congestions between the grids in f1 are more evenly distributed than that for f1. As a result, it means that the

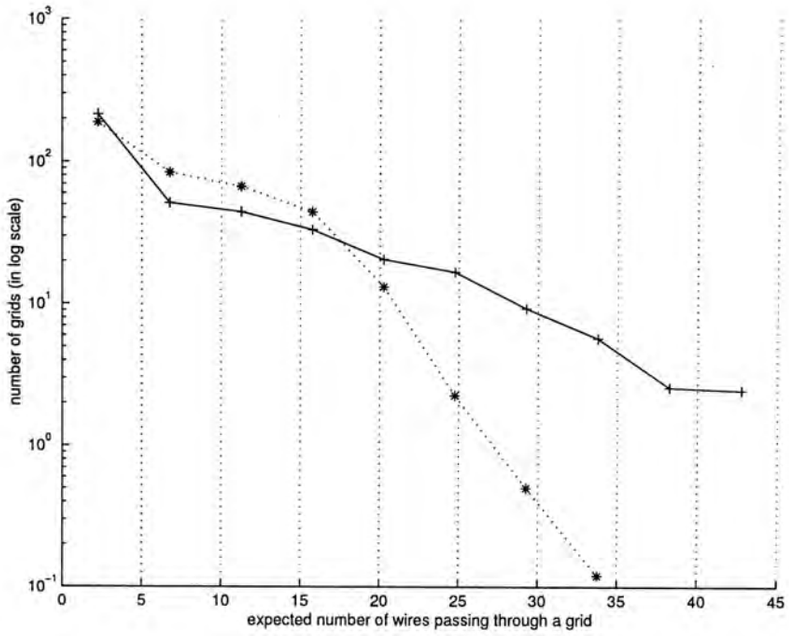


Figure 4.12: Distribution of wires among the grids for ami49

floorplanner fl can reduce the congestion in the most congested grids.

The statistical result is shown in table 4.2. We can see that the average congestion of the top 10% most congested grids and the number of blocked wires can be reduced significantly without a large penalty on area and wire-length. However, the run time of the single level floorplanner with congestion estimation is 10 to 20 times longer than that without congestion estimation.

4.9 Summary

In this chapter, we present a congestion model for estimating interconnect cost with fixed buffer planning in a routability-driven floorplanner. By using this probabilistic model, the number of blocked nets, delay and congestion can be estimated, taken into account buffer locations and routing blockages. Experimental results show that congestion and delay can be better optimized using this congestion model with no significant penalty in area.

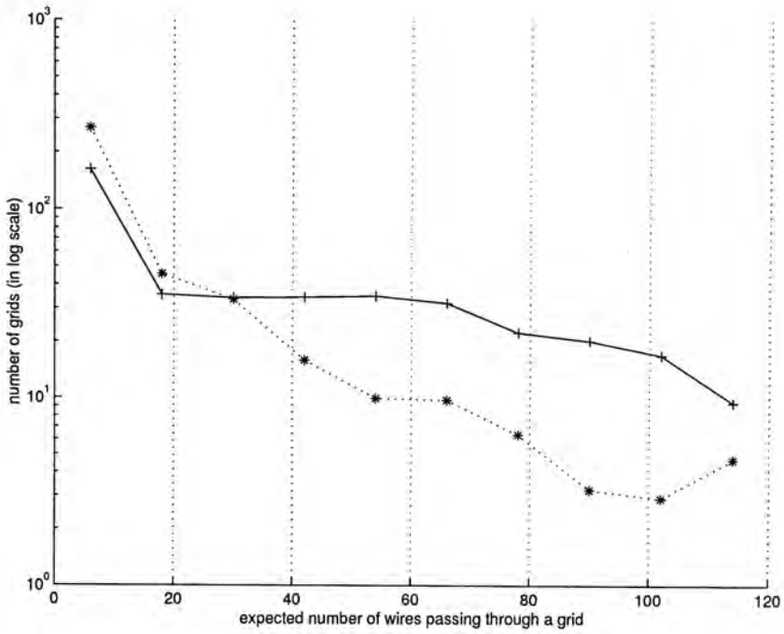


Figure 4.13: Distribution of wires among the grids for playout

Case	Floorplanner with congestion est.			Floorplanner without congestion est.		
	ami33	*ami49*	*playout*	*ami33*	*ami49*	*playout*
Area ($10^3\mu m^2$)	1245.8	39941.9	1000.1	1233.1	39415.6	969.2
Deadspace (%)	8.2	11.1	11.2	7.3	10.1	8.4
Wirelength ($10^3\mu m^2$)	59.5	908.0	554.1	57.2	860.0	488.1
Congestion [1]	2.8	0.2	28.9	3.0	0.3	37.6
No. of blocked wire	0.0	1.0	3.0	2.0	6.0	11.0
Runtime (s)	6222.3	10750.2	46265.2	247.9	921.6	2797.2

[1]Congestion - Average number of nets per $10^3\mu m^2$ for the top 10% most congested grids

Table 4.2: Comparison between the single level floorplanner with and without congestion estimation

Chapter 5

Floorplanner with Flexible Buffer Planning [35]

5.1 Introduction

Experimental results of the routability-driven floorplanner with fixed buffer planning show that congestion and delay can be better optimized using the congestion model discussed in chapter 4 with little penalty in area. However, the major disadvantages of this method are that it will take a long computational time to compute the exact locations of buffer insertions and buffers are not required to be inserted at fixed intervals along the routes in real cases. In this chapter, we propose a similar method to estimate congestion and routability but using a more realistic assumption on buffer insertions. Instead of computing the exact locations of the buffers, we assume that buffers are constrained to be inserted for long enough wires such that the distance between adjacent buffers is lying within a range $[low, up]$ given by the user. We call this the *variable interval buffer insertion constraint* [9, 2].

In this routability-driven floorplanner with flexible buffer planning, we will estimate congestion at each grid subject to the given variable interval buffer insertion constraint. Notice that the congestion at each location is dependent on the routes of the wires while the route of a wire is, in turn, dependent on

the availability of buffer resources along the route. This availability of buffer resources can be estimated from the amount of empty space at each location and the number of possible buffer insertions at that location. We compute the congestion information assuming that every route is equally likely to be used as long as the buffer insertion constraint can be satisfied. The computation can be performed efficiently by using dynamic programming and a table lookup approach. Besides, we have used a two stage simulated annealing method to speed up the whole floorplanning process.

5.2 Overview of the Floorplanner

In this chapter, we present the routability-driven floorplanner with buffer block planning in which variable buffer insertion constraint [35] is employed. We assume that the wires are routed over-the-cell and multi-bend routing is used. We divide a floorplan into grids and the deadspace in each grid is calculated for estimation of the amount of buffer resources.

In this floorplanner, given a candidate floorplan solution, we will first estimate the buffer usage of each wire k at each grid, then we can estimate the total buffer usage at each grid, $b_usage(x, y)$. We assume that buffers should be inserted at flexible intervals from each other for long enough wires. With the maximum number of buffer insertions allowed at each grid $b_space(x, y)$, ($b_space(x, y)$ is estimated from the amount of empty space of grid (x, y)) we can estimate the probability of successful buffer insertion $b_success(x, y)$. We can then make use of this information to estimate the congestion information $weight(x, y)$, taken into account the buffer insertion constraint. In order to improve the routability of the design, our approach will try to reduce congestion at different locations of the layout and to ensure that the nets can satisfy their delay constraints with proper buffer insertions using a two stage simulated annealing method.

5.3 Congestion Model

In order to improve the routability of the floorplan solution, a congestion model is used to estimate the congestion information. By using a good congestion model, the intermediate floorplan solution can be evaluated accurately according to their congestion and routability. The paper [20] shows that their probabilistic congestion model correlates well to post-route results but their work does not consider buffer locations. This motivates us to use a probabilistic model in our floorplanner. Nevertheless, as the clock frequencies reach and exceed the gigahertz level, each top level global net must undergo buffer insertion to maintain signal integrity and reasonable signal delay. Since the number of buffer insertions is large and buffers cannot be inserted just anywhere, buffer insertions can affect the final routing and thus the circuit performance significantly. Buffer planning in the early designing stage is very important and the rest of the design flow will be benefited by a proper planning and usage of the buffer resources.

A commonly used congestion model is shown in figure 5.1. We can see that the probability of each possible route for a wire k is equal. However, if the probabilities of successful buffer insertion (dependent on the amount of buffer resources) are different between the grids, the probabilities of some routes will be higher if they pass through the locations with higher probabilities of successful buffer insertion.

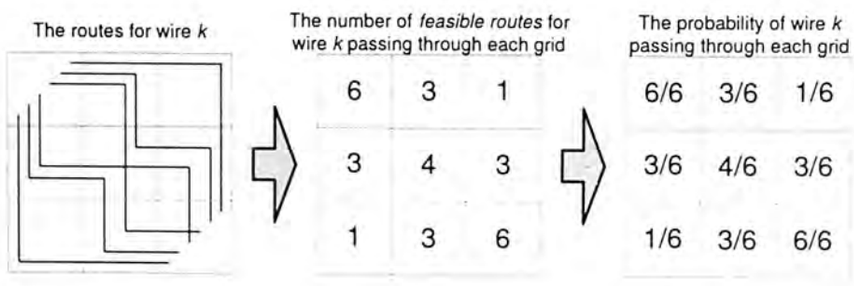


Figure 5.1: A commonly used congestion model

In our approach, we try to address the buffer insert issue in a new congestion model. In this model, the adjacent buffers are inserted at distance $[low, up]$ from each other where low and up are the lower and upper bound of the buffer insertion constraint respectively. An example is shown in figure 5.2. In this example, we assume that both the upper and lower bound in the buffer insertion constraint are equal to two, i.e., buffers should be inserted at an interval of two grid unit length from each other. We can see that the probabilities of the route b, c, d and e with successful buffer insertions are only 0.5. Therefore, the total number of *feasible routes* for this wire is reduced from six to four where the number of *feasible routes* is the summation of the probabilities of all possible routes without violating the buffer insertion constraint. As a result, the probability of wire k passing through each grid will be different from that computed in figure 5.1. The probabilities of some routes become higher if they pass through the locations with a higher probability of successful buffer insertion.

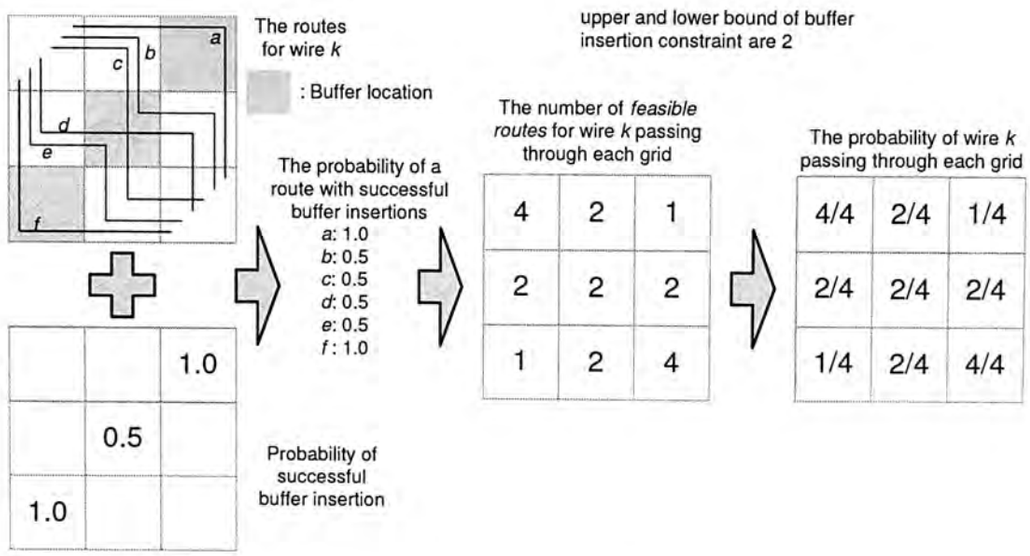


Figure 5.2: A simple example of our congestion model

In order to compute the congestion information using this model, we need

estimates of $b_usage(x, y)$ and the $b_success(x, y)$ at each grid (x, y). In this section, we will show the computation of the congestion information, assuming that $b_success(x, y)$ is given for each grid (x, y). We will show how we can estimate these buffer resources information in section 5.4.

5.3.1 Probabilistic Model with Variable Interval Buffer Insertion Constraint

Given the information on buffer resources at each grid, we want to calculate the probability $r_success(l)$ that a route l can be routed successfully from the source to the sink without violating any buffer insertion constraint. In the example of figure 5.3, there are only two feasible ways of buffer insertion for each route because the wirelength is six and the upper bound and lower bound of the buffer insertion constraint are three and two respectively. The probability of the first feasible way with all buffers inserted successfully is equal to $b_success(x_0 + 2, y_0) \times b_success(x_0 + 4, y_0)$, while that of the second one is equal to $b_success(x_0 + 3, y_0)$. Given the probabilities of successful buffer insertion, the probability of route 1 satisfying the buffer insertion constraint, $r_success(1)$, can be computed as $\frac{0.2+0.5}{2} = 0.35$. Similarly, we can calculate $r_success(l)$ for all the other routes $l = 2, 3, 4, 5, 6$.

Consequently, the total number of *feasible routes* of a wire k is equal to $\sum_{l \in L_k} r_success(l)$ where L_k is the set of all routes for wire k, and the number of *feasible routes* passing through the grid (x, y) is computed as follows:

$$\sum_{l \in L_k(x,y)} r_success(l)$$

where $L_k(x, y)$ is the set of all routes for wire k passing through the grid at (x, y). For the example in figure 5.3,

$$\begin{aligned} \sum_{l \in L_k} r_success(l) &= 3 \times 0.25 + 2 \times 0.35 + 0.1 \\ &= 1.55 \end{aligned}$$

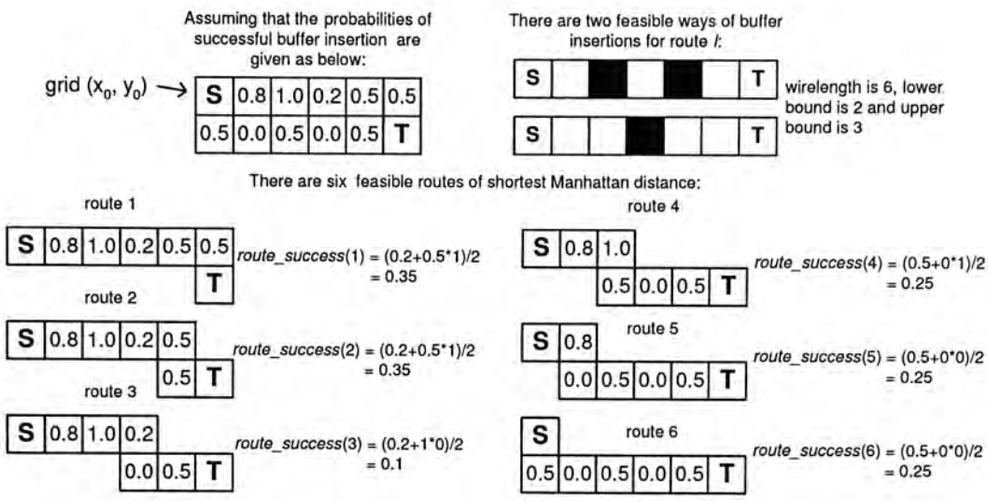


Figure 5.3: Example of computing $route_success(l)$

Considering the grid at (x_0+1, y_0), route 1 ,2, 3, 4 and 5 will pass through this grid, so the total number of *feasible routes* passing through this grid is equal to $2 \times 0.25 + 2 \times 0.35 + 0.1$, i.e., 1.3. Consider the grid at $(x_0, y_0 + 1)$, only route 6 will pass through this grid. Thus, the total number of *feasible routes* passing through this grid is equal to $r_success(6)$, i.e., 0.25.

Consequently, the congestion information $F(x, y, k)$ at grid (x, y) for wire k can be calculated as:

$$F(x, y, k) = \frac{\sum_{l \in L_k(x,y)} r_success(l)}{\sum_{l \in L_k} r_success(l)},$$

where $F(x, y, k)$ is the probability of wire k passing through grid (x, y). For example, we can compute $F(x_0 + 1, y_0, k) = \frac{1.3}{1.55}$ and $F(x_0, y_0 + 1, k) = \frac{0.25}{1.55}$. We can see that $F(x_0 + 1, y_0, k) + F(x_0, y_0 + 1, k)$ is equal to one because the routes of wire k must either pass through the grid at $(x_0 + 1, y_0)$ or at $(x_0, y_0 + 1)$. Finally, the average number of wires passing through a grid at (x, y) is computed as:

$$weight(x, y) = \sum_{all\ wire\ k} F(x, y, k)$$

In order to calculate $F(x, y, k)$ efficiently, we use dynamic programming. At the beginning, we are given the probability of successful buffer insertion at

each grid. We will then create an array $inf[0...up]$ at each grid with size equal to one plus the upper bound of the buffer insertion constraint, up. Assuming that wire k starts from grid (x_0, y_0) and ends at grid (x_t, y_t), we will initialize $g_0(x_0, y_0, k).inf[0]$ to one and the remaining values of the array to zero. The value of $g_0(x, y, k).inf[i]$ represents the total number of *feasible routes* from the source to (x, y) such that the last buffer is inserted at a grid of distance i units before (x, y). The values of the array at each grid can be computed dynamically row by row accordingly to the following recursive equation:

$$\begin{aligned} v_1 &= g_0(x-1, y, k).inf[i-1] + g_0(x, y-1, k).inf[i-1] \\ v_2 &= \textstyle\sum_{i=low}^{up} g_0(x, y, k).inf[i] \times b_success(x, y) \end{aligned}$$

$$g_0(x, y, k).inf[i] = \begin{cases} v_1 & \text{for } 1 \leq i \leq up \\ v_2 & \text{for } i = 0 \end{cases} \tag{5.1}$$

Consequently, the value $g_0(x_t, y_t, k).inf[0]$ can be obtained that represents the total number of *feasible routes* for wire k from the source (x_0, y_0) to the sink (x_t, y_t). Then, we will repeat the same steps from the sink (x_t, y_t) to the source (x_0, y_0) to calculate $g_1(x, y, k).inf[i]$'s by following equation:

$$\begin{aligned} v_3 &= g_1(x+1, y, k).inf[i-1] + g_1(x, y+1, k).inf[i-1] \\ v_4 &= \textstyle\sum_{i=low}^{up} g_1(x, y, k).inf[i] \times b_success(x, y) \end{aligned}$$

$$g_1(x, y, k).inf[i] = \begin{cases} v_3 & \text{for } 1 \leq i \leq up \\ v_4 & \text{for } i = 0 \end{cases} \tag{5.2}$$

Finally, the total number of feasible routes passing through the grid at (x, y) and a buffer is inserted at (x, y) can be computed as:

$$F_1(x, y, k) = \frac{g_0(x,y,k).inf[0] \times g_1(x,y,k).inf[0]}{b_success(x,y)}.$$

In addition, the total number of *feasible routes* passing through the grid (x, y) and no buffer is inserted at (x, y) can be computed as:

$$F_0(x, y, k) = \sum_{i,j} g_0(x, y, k).inf[i] \times g_1(x, y, k).inf[j]$$

$$\text{where } i, j \neq 0 \text{ and } low \leq i + j \leq up$$

Finally, as the total number of *feasible routes* passing from source (x_0, y_0) to sink (x_t, y_t) is $g_0(x_t, y_t, k).inf[0]$ or $g_1(x_0, y_0, k).inf[0]$ ($g_0(x_t, y_t, k).inf[0]$ must equal $g_1(x_0, y_0, k).inf[0]$), the probability that wire k will pass through the grid at (x, y) can be computed as:

$$F(x, y, k) = (F_0(x, y, k) + F_1(x, y, k))/g_0(x_t, y_t, k).inf[0]$$

$g_0(x_0,y_0,k).inf[3]\rightarrow$	0.00	0.00	0.00	1.00	0.00	1.00
$g_0(x_0,y_0,k).inf[2]\rightarrow$	0.00	0.00	1.00	0.00	1.00	0.20
$g_0(x_0,y_0,k).inf[1]\rightarrow$	0.00	1.00	0.00	1.00	0.20	0.50
$g_0(x_0,y_0,k).inf[0]\rightarrow$	1.00	0.00	1.00	0.20	0.50	0.60
	0.00	0.00	3.00	0.00	3.00	2.10
	0.00	2.00	0.00	2.00	1.90	1.00
	1.00	0.00	1.00	1.70	0.50	3.05
	0.00	0.00	1.50	0.00	2.45	3.10

Figure 5.4: Result of $g_0(x, y, k).inf[i]$'s

$g_1(x_0,y_0,k).inf[3]\rightarrow$	2.10	1.00	0.00	3.00	0.00	0.00
$g_1(x_0,y_0,k).inf[2]\rightarrow$	1.00	1.60	1.00	0.00	2.00	0.00
$g_1(x_0,y_0,k).inf[1]\rightarrow$	2.33	1.00	1.10	1.00	0.00	1.00
$g_1(x_0,y_0,k).inf[0]\rightarrow$	3.10	2.08	1.00	0.60	1.00	0.00
	0.00	0.00	1.00	0.00	0.00	0.00
	0.50	0.00	0.00	1.00	0.00	0.00
	0.00	0.50	0.00	0.00	1.00	0.00
	0.25	0.00	0.50	0.00	0.00	1.00

Figure 5.5: Result of $g_1(x, y, k).inf[i]$'s

Consider the case in figure 5.3 again, we can compute $g_0(x, y, k).inf[i]$'s and $g_1(x, y, k).inf[i]$'s by equation 5.1 and equation 5.2. The results are shown in figure 5.4 and figure 5.5. In this case, $F(x_0, y_0 + 1, k)$ can be calculated as

follows:

$$\begin{aligned}
F_0(x_0+1,y_0,k) &= g_0(x_0+1,y_0,k).inf[1] \times g_1(x_0+1,y_0,k).inf[1] + \\
&\quad g_0(x_0+1,y_0,k).inf[1] \times g_1(x_0+1,y_0,k).inf[2] + \\
&\quad g_0(x_0+1,y_0,k).inf[2] \times g_1(x_0+1,y_0,k).inf[1] \\
&= 1.0 \times 1.0 + 1.0 \times 1.6 + 1.0 \times 0.0 \\
&= 2.6 \\
F_1(x_0+1,y_0,k) &= \frac{t_0.inf[0] \times t_1.inf[0]}{0.8} \\
&= 0.0 \\
F(x_0+1,y_0,k) &= \frac{F_0(x_0+1,y_0,k)+F_1(x_0+1,y_0,k)}{g_0(x_t,y_t,k).inf[0]} \\
&= \frac{2.6+0.0}{3.1} \\
&= \frac{1.3}{1.55}
\end{aligned}$$

From the example, we can see that the result of $F(x_0+1,y_0,k)$ obtained by dynamic programming is the same as counting the route from figure 5.3. As a result, we can compute $F(x_0+1,y_0,k)$ effectively.

5.3.2 Time Complexity

According to this congestion estimation model, we need to scan the array at each grid from the source to the destination for each net. Therefore, the time complexity is $O(m \times k \times up)$ where k is number of grids scanned for each net, m is number of nets and up is the upper bound of the buffer insertion constraint. If the floorplan is divided into $K \times K$ grids, we need to scan at most K^2 grids for each net. Moreover the upper bound in the buffer insertion constraint is also bounded by the length of the wires. (It is bounded by $2K$). Therefore, the time complexity of this congestion estimation is only $O(m \times K^3)$.

5.4 Buffer Planning

In this section, we will show how we can estimate the buffer usage at each grid (x, y), $b_usage(x, y)$ and calculate the probability of successful buffer insertion at each grid (x, y), $b_success(x, y)$. Such information will be used in calculation of the congestion information that we have discussed it in section 5.3.1.

5.4.1 Estimation of Buffer Usage

According to the flexible interval buffer insertion constraint, buffers can be inserted at flexible locations as long as adjacent buffers are at a distance $x \in [low, up]$ from each other. There are thus several feasible ways of buffer insertion satisfying the constraint for each route. The probability that a route l of wire k will insert a buffer at grid (x, y), $b_insert(x, y, l, k)$, should be calculated in order to estimate the amount of buffer usage, $b_usage(x, y)$. Given a possible route l of wire k with source S and sink T and the buffer insertion constraint $[up, low]$, the total number of feasible ways of buffer insertions *total* can be obtained. We assume that all feasible ways of buffer insertions have the same probability of being used in the routing stage. The probability that a grid (x, y) is required to have a buffer inserted for a route l of wire k can then be calculated as follows.

$$b_insert(x, y, l, k) = \frac{N(dist)}{total},$$

where *dist* is the Manhattan distance of the grid (x, y) from the source S, *total* is the total number of feasible ways of buffer insertions for l, $N(dist)$ is the number of feasible ways of buffer insertions for route l that will insert a buffer at a grid of distance *dist* from the source and $b_insert(x, y, l, k)$ is the probability that the route l will insert a buffer at grid (x, y). An example is shown in figure 5.6.

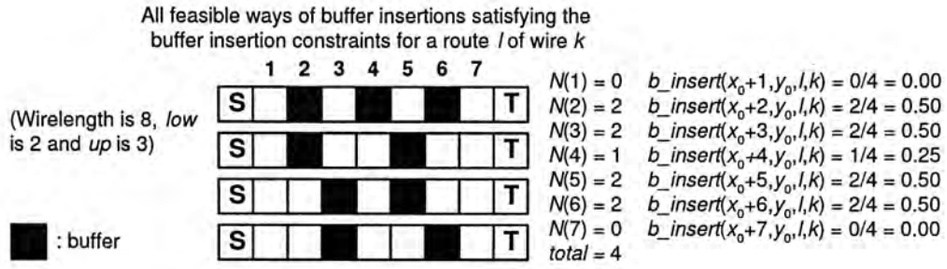


Figure 5.6: Example of calculating the probabilities of successful buffer insertion at each grid

In the example of figure 5.6, there are four feasible ways of buffer insertions satisfying the buffer insertion constraints, so *total* is equal to four. Consider the grid $(x_0 + 4, y_0)$, its distance from the source S is equal to four, so *dist* is four. There is only one feasible way of buffer insertion that will insert a buffer at a grid of distance four from the source, so $N(4)$ is equal to one and the probability that grid $(x_0 + 4, y_0)$ is required to have a buffer inserted can be calculated as follows:

$$\begin{aligned} b_insert(x_0 + 4, y_0, l, k) &= \frac{N(4)}{4} \\ &= \frac{1}{4} \end{aligned}$$

We can compute these $b_insert(x, y, l, k)$ from $cnt(D, up, low)$ where $cnt(D, up, low)$ is the total number of feasible ways of buffer insertion from a grid A to another grid B such that D is the unit grid distance between A and B, (note that either there is a buffer which is inserted at both A and B or they are either source or sink) and *up* and *low* are the bounds in the buffer insertion constraint. The probability of successful buffer insertion at a grid (x, y) of distance *dist* from the source can be calculated as:

$$b_insert(x, y, l, k) = \frac{cnt(dist,up,low) \times cnt(length-dist,up,low)}{cnt(length,up,low)},$$

where *length* is the length of route l, $cnt(dist, up, low)$ is the number of feasible ways of buffer insertion from the source to grid (x, y), $cnt(length-dist, up, low)$ is the number of feasible ways of buffer insertion from the grid (x, y) to the sink and $cnt(length, up, low)$ is the number of feasible ways of buffer insertion from the source to the sink. In order to compute the probability of buffer insertion, $b_insert(x, y, l, k)$, at each grid (x, y) for every route l and wire k efficiently, two methods based on dynamic programming are used to calculate $cnt(D, up, low)$. In both methods, $cnt(D, up, low)$ can be calculated, saved and reused. Such two methods are called the forward recursive method and the backward recursive method.

Forward Recursive Method to Compute $cnt(D, up, low)$

The forward recursive method works by searching and counting the number of feasible ways of buffer insertions that satisfies the buffer insertion constraint. For each successful way of buffer insertion found, the number accumulated will be increased by one. Finally, the total number of feasible ways of buffer insertions between two points of distance D, $cnt(D, up, low)$, will be obtained. The algorithm is shown in figure 5.7 and an example is shown in figure 5.8.

For the example in figure 5.8, D is equal to eight and the upper and lower bound of the buffer insertion constraint are three and two respectively, so $up = 3$ and $low = 2$. At the beginning, the function of $cnt(8, 3, 2)$ is called. We will then try to insert the first buffer at the grid $(x_0 + 2, y_0)$ that is at a distance low from the source S. The function $cnt(6, 3, 2)$ is then called recursively. (Note that after the return of $cnt(6, 3, 2)$, $cnt(5, 3, 2)$ will be called because we can also insert the first buffer at the grid $(x_0 + 3, y_0)$ that is at a distance three from the source S.) When $cnt(6, 3, 2)$ is called, we will try to insert the second buffer at the grid $(x_0 + 4, y_0)$ that is at a distance low from the first buffer. The function $cnt(4, 3, 2)$ is then called recursively again. This process will repeat until $cnt(2, 3, 2)$ is called and "1" will be returned because

Forward recursive method: $cnt(D, up, low)$

Input: Distance between the two target grids, D
Lower bound of buffer insertion constraint, low
Upper bound of buffer insertion constraint, up
Output: Total number of feasible ways of buffer insertions satisfying the buffer insertion constraint, $total$

$total = 0$
$up_limit = min(up, D)$
for $(i = low; i \leq up_limit; i++)$
  if $D - i = 0$
    $total = total + 1$
  else if $D - i > 0$
    $total = total + cnt(D - i, up, low)$
return $total$

Figure 5.7: The algorithm of Forward recursive method

the final value of D ($D = 2$) is lying between low and up. In general, the function $cnt(D, up, low)$ will be called recursively until all the feasible ways of buffer insertions are found. The formula is shown below:

$$cnt(D, up, low) = \begin{cases} \sum_{i=low}^{up} g_1(D, i, up, low) & \text{if } D \geq low \\ 0 & \text{if } D < low, \end{cases}$$

$$\text{where } g_1(D, i, up, low) = \begin{cases} cnt(D - i, up, low) & \text{if } D > i \\ 1 & \text{if } D = i \\ 0 & \text{if } D < i \end{cases}$$

According to forward recursive method, the number of recursive calls, r_1 can be calculated as follows:

$$\begin{aligned} r_1 &= 1 + D - 1 + (D - 2) + \cdots + (D - up) \\ &= 1 + \frac{up}{2}(2D - up - 1) \end{aligned}$$

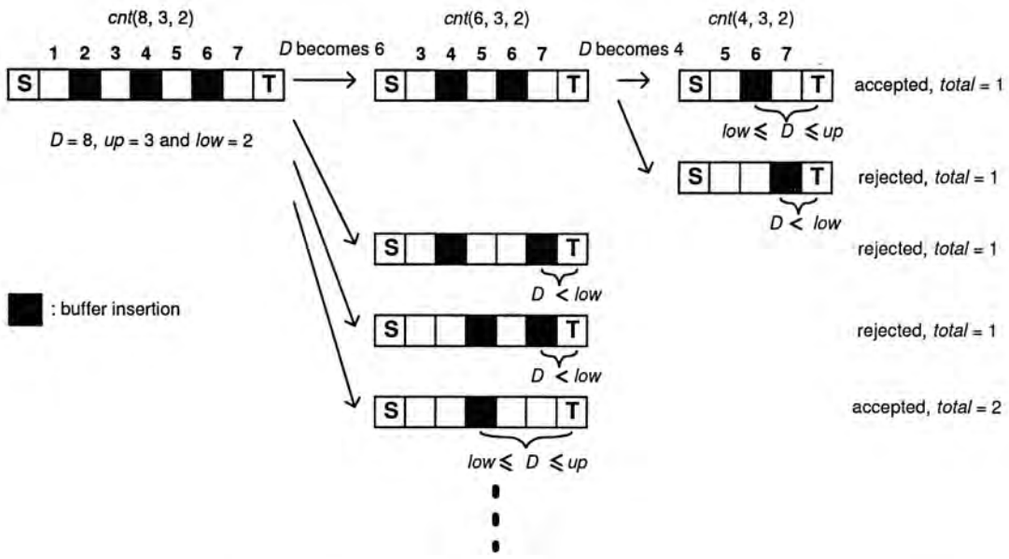


Figure 5.8: Example of the forward recursive method

Backward Recursive Method to Compute $cnt(D, up, 1)$

The backward recursive is a faster method but it has a limitation that the lower bound of the buffer insertion constraint must be one. It counts the number of infeasible ways of buffer insertions by looking at the first position where a violation occurs. Finally, it can find out the total number of infeasible ways of buffer insertions, $fail_total$. Since the total number of ways for buffer insertions is 2^{D-1}, the total number of feasible ways of buffer insertion satisfying the buffer insertion constraint, $total$, can be calculated as $2^{D-1} - fail_total$. The detail algorithm is shown in figure 5.9 and an example is shown in figure 5.10.

For the example in figure 5.10, D is equal to eight and the upper and lower bound of the buffer insertion constraint are three and one respectively, so $up = 3$ and $low = 1$. At the beginning, $cnt(8, 3, 1)$ is called. We will first count the number of ways of buffer insertion such that the first violation occurs at the leftmost possible grid. This occurs when the first buffer is not inserted until the grid $(x_0 + 4, y_0)$ because the interval between the source S and the first buffer is greater than up and the buffer constraint cannot be satisfied. All

Backward recursive method: $cnt(D, up, 1)$

Input: Distance between the two target grids, D
Upper bound of buffer insertion constraint, up
Output: Total number of possible ways of buffer insertions satisfying the buffer insertion constraint, *total*

$total = 2^{D-1}$
for $(i = 0; i \leq D - up - 1; i++)$
  $total = total - 2^{D-up-i-1} * cnt(i, up, 1)$
return *total*

Figure 5.9: The algorithm of backward recursive method

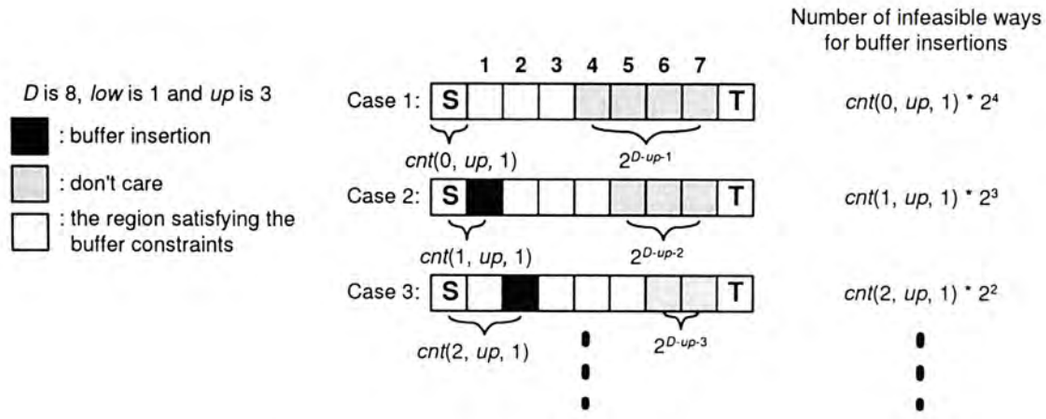


Figure 5.10: Backward recursive method

the ways that insert the first buffer after the grid $(x_0 + 3, y_0)$ are infeasible. As a result, the total number of infeasible ways of buffer insertion in this case is equal to 2^{D-up-1}, i.e., 2^4. Then if the first violation occurs at the grid $(x_0 + 5, y_0)$, the total number of infeasible ways of buffer insertion in this case is equal to $cnt(1, up, 1) \times 2^{D-up-2}$, i.e., $cnt(1, up, 1) \times 2^3$. Notice that since the first violation occurs at the grid $(x_0 + 5, y_0)$, those insertions in front must be correct and we have made use of the recursive call $cnt(1, up, 1)$ to count those feasible insertions in front. In this way, the function will be called recursively

to obtain the result $cnt(D, up, 1)$. The general formula is shown below:

$$cnt(D, up, 1) = \begin{cases} 0 & \text{if } D = 0 \\ g_2(D, up) & \text{if } D > 0, \end{cases}$$

$$\text{where } g_2(D, up) = 2^{D-1} - \sum_{i=0}^{D-up-1} 2^{D-i-up-1} \times cnt(i, up, 1)$$

According to the backward recursive method, the number of recursive calls, r_2 can be calculated as:

$$\begin{aligned} r_2 &= 1 + (D - up - 1) + (1 + \cdots + (D - 2up - 2)) \\ &= D - up + \frac{(D - 2up - 1)(D - 2up - 2)}{2} \end{aligned}$$

Assuming that the time required for each recursive call is similar for the two methods, we find that that the backward recursive method is faster when $up > \frac{D}{5}$:

$$\begin{aligned} r_2 - r_1 &< 0 \\ (1 + (D - up - 1) + (1 + \cdots + (D - 2up - 1))) - & \\ (1 + (D - 1) + (D - 2) + \cdots + (D - up)) &< 0 \end{aligned}$$

$$\begin{aligned} \frac{(D - 5up)(D - up - 1)}{2} &< 0 \\ D - 5up &< 0 \\ up &> \frac{D}{5} \end{aligned}$$

As a result, we can combine the two methods to find the total number of feasible ways of buffer insertions satisfying the buffer constraints efficiently. The algorithm is shown in figure 5.11.

Combined method: $cnt(D, up, low)$

Input: Distance between the two target grids, D
Lower bound of buffer insertion constraint, low
Upper bound of buffer insertion constraint, up
Output: Total number of feasible ways satisfying the buffer insertion constraint, *total*

if $(up > \frac{D}{5})$ and $(low = 1)$
  total = backward recursive method
else
  total = forward recursive method
return *total*

Figure 5.11: The algorithm of combined method

Finally, given the distance ($length$) of a route l of a wire k running from a source S to a sink T and the distance ($dist$) from the source to a grid (x, y). We can compute the total number of feasible ways of buffer insertions from S to T such that a buffer is inserted at (x, y) as:

$$cnt(dist, up, low) \times cnt(length - dist, up, low)$$

Therefore, the probability that a buffer is required at grid (x, y) for route l of wire k is:

$$b_insert(x, y, l, k) = \frac{cnt(dist,up,low)*cnt(length-dist,up,low)}{cnt(length,up,low)}$$

Note that these values can be computed, saved and reused for the wires of the same length.

5.4.2 Estimation of Buffer Resources

In order to obtain the congestion information, we need to compute the probability of successful buffer insertion at each grid. We have shown in the above section the computation of the probability that a route l of wire k will insert

a buffer at a grid (x, y), i.e., $b_insert(x, y, l, k)$, the total number of buffer insertions at grid (x, y), $b_usage(x, y)$, can then be calculated as:

$$b_usage(x, y) = \sum_{all\ wire\ k\ and\ all\ route\ l} b_insert(x, y, l, k)$$

As a result, the probability of successful buffer insertion at grid (x, y), $b_success(x, y)$, can be calculated as:

$$b_success(x, y) = min(1, \frac{b_space(x,y)}{b_usage(x,y)}),$$

where $b_space(x, y)$ is the number of buffers that can be inserted at grid (x, y) which is dependent on the amount of empty space in grid (x, y). From the equation, we can see that if the amount of the empty space of the grid (x, y) is large enough to accommodate for all possible buffer insertions, the value of $b_success(x, y)$ is equal to one. Otherwise, it is equal to $b_space(x, y)/b_usage(x, y)$.

5.5 Two-phases Simulated Annealing

Simulated annealing is an iterative and non-deterministic optimization technique. In simulated annealing, all the moves that result in a decrease in cost will be accepted. The moves that result in an increase in cost are accepted with a probability that decreases over the iterations. The algorithm is shown in figure 5.12.

In this floorplanner, the simulated annealing process is divided into two phases. They are the area optimization phase and the congestion optimization phase. In the area optimization phase, the congestion information is less meaningful because the locations of the modules are still far from its final position. Thus, the cost function used in this area optimization phase does not include the congestion cost and buffer insertion cost. The cost function is shown below:

$$Cost_0 = Area + \alpha \times Wire,$$

Simulated Annealing

temp = *init_temperture*
place = *init_placement*
while *temp* > *final_temperature* do
  while *inner_loop_criterion* = FALSE do
    new_place =PERTURB(*place*)
    $\triangle C$ =COST(*new_place*)−COST(*place*))
    if ($\triangle C < 0$) and RANDOM(0,1) > $e^{\frac{-\triangle C}{temp}}$
      place = *new_place*
    temp =SCHEDULE(*temp*)

Figure 5.12: Simulated Annealing

where *Area* is the area of the floorplan and $Wire$ is the total wirelength and α is a parameter.

In the congestion optimization phase, the intermediate solutions are close to the final packing. Therefore, the congestion information is meaningful and should be included. The cost function is shown below.

$$Cost_1 = Area + \alpha \times Wire + \beta \times M_weight,$$

where *Area* is the area of the floorplan, $Wire$ is the total wirelength (it will be discussed in section 5.6), M_weight is the average number of wires in the top ten percent most congested grids and α and β are parameters.

In the transitional period from the area optimization phase to the congestion optimization phase, we need to re-calculate the temperature because the order of magnitude of the cost may change a lot and the acceptance rate will drop or rise unexpectedly if we do not adjust the temperature. As a result, we need to use a new temperature to maintain the acceptance rate. To achieve this, we will first obtain the mean value of $\triangle Cost$ from a number of random walks on changing the packings before the phase change, and we will then obtain the mean value of $\triangle new_Cost$ from a number of random walks

on changing the packings using the new cost function after the phase changes. The new temperature *temp* can be computed as below:

$$temp = \frac{\triangle new_Cost}{\triangle Cost} \times old_temp$$

By using *temp*, the acceptance rate can be maintained and the change of the cost function can be performed smoothly.

5.6 Wirelength Estimation

In this stage, the positions of the I/O pins in the modules are not yet fixed and the modules will usually cover a number of grids. Therefore, the assumption on the locations of the I/O pins will affect the results significantly in our method. Consequently, we need to estimate reasonably the positions of the I/O pins in order to compute the interconnect costs and congestion cost accurately. We have discussed it in more detail in section 4.6 of chapter 4.

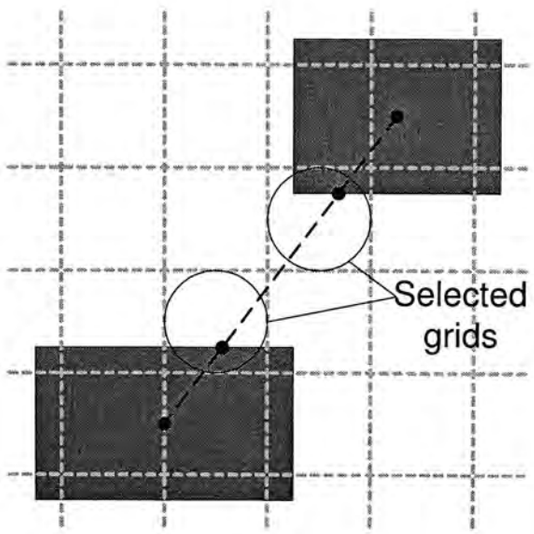


Figure 5.13: Intersection-to-intersection estimation

In order to distribute the I/O pins into the grids appropriately, intersection-to-intersection method is (figure 5.13) is used. This is an appropriate method

for our floorplanner since the estimated positions of the I/O pins will be similar to those in the final design and the buffer locations can be estimated more accurately. However, extra calculations are needed which may lead to slower speed.

5.7 Multi-pin Nets Handling

In our design, congestion estimation with flexible buffer block planning can be applied when all the nets are two-pin wires only. In order to handle multi-pin nets, we need to decompose the multi-pin nets into a set of two-pin wires by using minimum spanning tree (MST). We have discussed it in more detail in section 4.7 of chapter 4.

5.8 Experimental Results

	*ami*33	*ami*49	*playout*
[1] Grid length (μm)	60	400	50
Enlargement	10	1	10
[2] Wiring Capacity	6	18	60
up	3	2	3
low	6	4	6

[1] Before enlargement

[2] Wiring Capacity is the maximum number of wire that can pass through each grid

Table 5.1: Configuration of the experiments

We have implemented three floorplanners, a traditional floorplanner $F1$ based on simulated annealing without considering congestion and buffer planning, a routability-driven floorplanner $F2$ based on simulated annealing using the probabilistic model for buffer planning with variable interval buffer insertion constraint, and another routability-driven floorplanner $F3$ that is similar

Cases	*ami*33		*ami*49		*playout*	
	$F1$	$F3$	$F1$	$F3$	$F1$	$F3$
[3] Area ($10^3\mu m^2$)	1293.44	1296.61	39454.41	39754.35	990.58	992.54
[3] Wirelength ($10^3\mu m$)	20.64	20.59	399.75	379.80	306.76	275.55
[1] Congestion	3.07	2.13	0.08	0.08	19.61	18.48
No. of unroutable wires	14.75	9.63	13.13	9.88	163.88	115.88
[2] Runtime (s)	127.71	1333.90	151.56	3573.79	494.45	11680.80

[1] Average number of nets per $10^3\mu m^2$ for the top 10% most congested grids
[2] Experiments are performed using Pentium 4 1.2GHz with 512Mb memory
[3] Before enlargement

Table 5.2: Comparison between $F1$ and $F3$

to $F2$ except that two-phases simulated annealing is used. The data sets used in the experiments are *ami*33, *ami*49 and *playout*. The configuration of the experiments is shown in table 5.1. The buffer insertion constraint $[low, up]$ used are [3,6], [2,4] and [3,6] respectively in terms of grid lengths (approximately equal to $[2100\mu m, 4200\mu m]$). The value of low and up are computed as below:

$$low = \frac{\sqrt{\frac{4*(R_b*C_b+D_b)}{R_o*C_o}}}{(2*10*g_unit)},$$

$$up = \frac{\sqrt{\frac{4*(R_b*C_b+D_b)}{R_o*C_o}}}{(10*g_unit)}$$

g_unit is the length of a grid and the others are the physical parameters in the $0.18\mu m$ technology. The above equations are optimized from the equation 4.1 by assuming $R_D = R_B$ and $C_L = C_B$ (The detail step is done on p.76 of in [44]). Notice that the sizes of the modules are enlarged for demonstration of the effect of buffer block planning. We have implemented a simple global router (we will discuss it in detail in chapter 6) to evaluate the performance of

Cases	*ami*33		*ami*49		*playout*	
	$F2$	$F3$	$F2$	$F3$	$F2$	$F3$
[3] Area ($10^3\mu m^2$)	1336.61	1296.61	39921.69	39754.35	1008.87	992.54
[3] Wirelength ($10^3\mu m$)	22.73	20.59	408.25	379.80	296.56	275.55
[1] Congestion	2.16	2.13	0.08	0.08	17.76	18.48
No. of unroutable wires	9.00	9.63	10.00	9.88	115.88	115.88
[2] Runtime (s)	2124.00	1333.90	9805.00	3573.79	54541.00	11680.80

[1] Average number of nets per $10^3\mu m^2$ for the top 10% most congested grids
[2] Experiments are performed using Pentium 4 1.2GHz with 512Mb memory
[3] Before enlargement

Table 5.3: Comparison between $F3$ and $F2$

the floorplanners. In the router, the nets will be routed one after another and a wire is unroutable when it cannot be routed form the source to the sink in the shortest Manhattan distance and satisfying the buffer and congestion (we assume that there is limitation on the number of wires in each grid) constraints.

Comparing $F1$ and $F3$ in table 5.2, we can see that the differences between $F1$ and $F3$ on area are very small but $F3$ can reduce the total wirelength, congestion and number of unroutable wires significantly. It demonstrates the effectiveness of our probabilistic congestion model in reducing the interconnect cost in floorplan design. Notice that the area penalty of $F3$ is very small and the running time is only double that of $F1$.

Table 5.3 shows the comparison between $F2$ and $F3$. We can see that the performance of $F2$ and $F3$ are similar except that the runtime of $F3$ is much smaller. This demonstrates that the two-phases simulated annealing approach can reduce the runtime of the whole floorplanning process significantly without any degradation in performance.

5.9 Remarks

In this floorplanner, all routes have equal chance being selected. However, simpler routing patterns are more commonly used in real cases. It means that there are more routes that have fewer bends. In our design, we can consider this condition by applying dynamic programming.

5.10 Summary

In this chapter, we present a new congestion model to compute the congestion information taking into account buffer planning with variable interval buffer insertion constraint. The estimations are based on the supply and demand analysis of routing and buffer resources, where the supply is determined by the floorplan solution (the amount of empty space) and the demand is determined by the interconnect structure. Computations of the congestion information in our model are quite complicate but they can be performed efficiently by dynamic programming. Experimental results show that our floorplanner can reduce the interconnect cost efficiently without much penalty in area. In addition, the runtime can be improved significantly without degradation in performance by the two-phases simulated annealing process.

Chapter 6

Global Router

6.1 Introduction

In order to evaluate the congestion and routability of the floorplanners in chapter 5, we have implemented a simple global router by using dynamic programming. In this router, the wire are routed based on the following criteria: congestion of each grid and number of buffers. We will discuss this router in detail in this chapter.

6.2 Overview of the Global Router

In this section, we will give a brief overview of the simple global router. First, the floorplan will be divided into a grid structure. The nets will then be decomposed into two-pin wires and the wires will be routed one after another. Each wire will be routed once only. If a wire can be routed from its source to its sink in the shortest Manhattan distance without violating the congestion constraint and the buffer insertion constraint, the wire is said to be routable. Otherwise, it is unroutable.

6.3 Buffer Insertion Constraint and Congestion Constraint

In our design, we assume that there is a limitation on the number of wires in each grid. We need to apply this congestion constraint because the routing resources in each grid are limited in the real cases. As a result, a wire can pass through a grid only if the number of wires in that grid has not reached the maximum and the congestion constraint can be satisfied. It means that a wire can pass through a grid only if there is enough routing resources.

In addition, the wires should also satisfy the buffer insertion constraint. In order to satisfy the timing constraint, we can insert buffers to reduce the fanout delay. However, buffer insertions may also increase the net delay. As a result, the number of buffer insertions and their locations should be controlled. In our buffer insertion constraint, adjacent buffers are required to be inserted at intervals of distance d where d is between a lower bound low and an upper bound up in order to satisfy the time constraint. The values of low and up are estimated analytically as below:

$$low = \frac{\sqrt{\frac{4*(R_b*C_b+D_b)}{R_o*C_o}}}{(2*10*g_unit)},$$

$$up = \frac{\sqrt{\frac{4*(R_b*C_b+D_b)}{R_o*C_o}}}{(10*g_unit)}$$

where g_unit is the length of a grid and the others are the physical parameters in the $0.18\mu m$ technology. R_b is the buffer resistance, C_b is the buffer capacitance, D_b is the intrinsic delay of a buffer, R_o is the unit wire resistance and C_o is the unit wire capacitance. The above equations are optimized from the equation 4.1 by assuming $R_D = R_B$ and $C_L = C_B$ (The detail step is done on p.76 of in [44]).

6.4 Multi-pin Nets Handling

In our router, we hope to perform global routing in an efficient way. The router perform routing by using minimum spanning tree (MST). Rectilinear steiner tree (RST) is the most commonly algorithm used in global routing but its run time is much slower than minimum spanning tree. MST may over-estimate the congestion because of the overlapping net segments, However, this conservative estimation will not affect the routing result significantly. According to the paper [11], the total length of an MST can be reduced at most by 6% to 9% only by removing all the overlapping net segments to get a corresponding RST.

6.5 Routing Methodology

During the routing process, we will find the best route for each wire based on the following criteria in decreasing order of priority:

- The route passes through less congested grids.
- The route requires less buffers to be inserted.
- The route passes through grids with more empty space such that more buffers can be inserted for other wires.

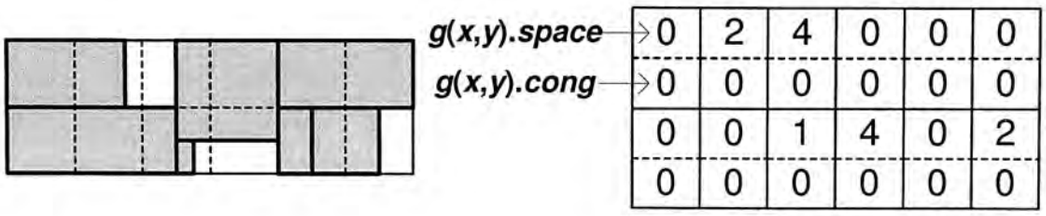


Figure 6.1: Construction of table with information of congestion and empty space

When a floorplan is given, we first divide the floorplan into a grid structure. A table is then constructed that contains the congestion information and the

number of buffer insertions allowed at each grid. An example is shown in figure 6.1. $g(x, y).cong$ is the congestion of the grid (x, y) and $g(x, y).space$ is the maximum number of buffers that can be inserted at the grid (x, y).

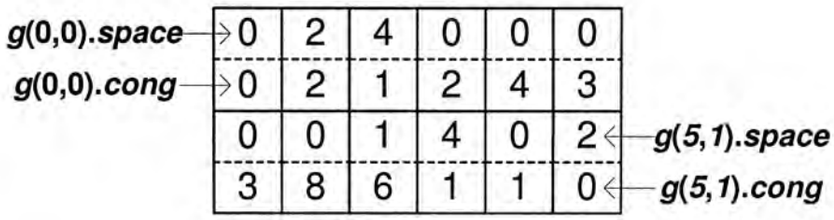


Figure 6.2: Updated information of congestion and empty space

After a wire is routed, the $g(x, y).cong$'s and $g(x, y).space$'s will be updated accordingly. Consider the example in figure 6.2 in which a wire is required to be routed from $(0, 0)$ to $(5, 1)$. We assume that the lower bound of the buffer insertion constraint is 2 and the upper bound is 3. In order to satisfy the buffer insertion constraint, buffers should be either inserted at the grid $(2, 1)$ or at grids $(2, 0)$ and $(3, 1)$. As a result, there are five possible routes and they are shown in figure 6.3.

From such routes, we will select the route such that the maximum congestion on that route is minimum. Therefore, route 4 will be selected though it requires two buffer insertions.

6.6 Implementation

In order to implement the global router based on the routing methodology in section 6.5, we apply dynamic programming. When a wire w is required to be routed from the grid (x_0, y_0) to the grid (x_t, y_t) where we assume that x_t is greater than x_0 and y_t is greater than y_0. If w should be routed in the shortest Manhattan distance, it must pass through either the grid $(x_t - 1, y_t)$ or $(x_t, y_t - 1)$. If the route passing from the grid (x_0, y_0) to the grid $(x_t - 1, y_t)$

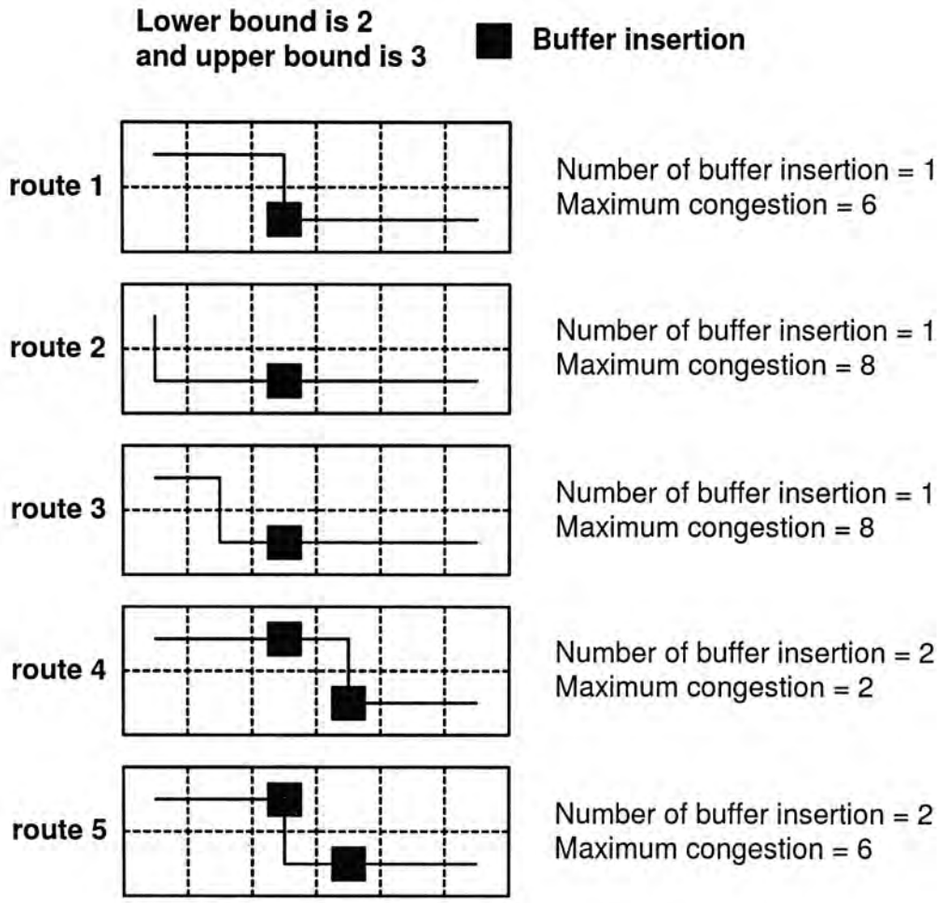


Figure 6.3: Different possible routes

is better than the route passing from the grid (x_0, y_0) to the grid $(x_t, y_t - 1)$ based on our routing methodology, w will pass through $(x_t - 1, y_t)$ rather than $(x_t, y_t - 1)$. We can thus choose the better one between these two and it will form part of the best route from (x_0, y_0) to (x_t, y_t). Similarly, we can find the best routes dynamically row by row by comparing the routes from the grid (x_0, y_0) to the grid at its top and to the grid on its left. The pseudo code is shown in figure 6.4.

Assuming that a wire starts from the grid (x_0, y_0) and ends at the grid (x_t, y_t) and x_t is greater than x_0 and y_t is greater than y_0. At the beginning, we will create an array $g(x, y).inf[i]$ for $0 \leq i \leq up$ at each grid (x, y) bounded by the grid (x_0, y_0) and the grid (x_t, y_t) where up is the upper bound

Global Router

Input: Congestion information at each grid $g(x, y).cong$
Empty Space at each grid, $g(x, y).space$
Source of wire, $source.x, source.y$
Sink of wire, $sink.x, sink.y$
Buffer insertion constraint, $lower, upper$
Output: Final route, $g(x, y).inf[0]$

```
for source.x ≤ i ≤ sink.x
  for source.y ≤ j ≤ sink.y
    for 1 ≤ k ≤ upper
      if i = source.x and j = source.y
        initialize g(x,y).inf[k]
      else
        temp = compare(g(x,y-1).inf[k-1], g(x-1,y).inf[k-1])
        (temp is the better route from (x,y-1) and (x-1,y))
          g(x,y).inf[k-1] = temp.inf
      if (i,j) ≠ (source.x, source.y) and (i,j) ≠ (sink.x, sink.y)
        g(x,y).inf[k].path = g(x,y).inf[k].path + temp.direction
        g(x,y).inf[k].cong = max(g(x,y).inf[k].cong, g(x,y).cong)
        if k ≥ lower
          g(x,y).inf[0] = compare(g(x,y).inf[k], g(x,y).inf[0])
    if (i,j) ≠ (source.x, source.y) and (i,j) ≠ (sink.x, sink.y)
      g(x,y).inf[0].buf_no = g(x,y).inf[0].buf_no + 1
      g(x,y).inf[0].buf = g(x,y).inf[0].buf +' 1'
```

Figure 6.4: Global Router

of the buffer insertion constraint. We can then store the information of the best route passing from the grid (x_0, y_0) to the grid (x, y) such that the previous buffer is inserted into a grid that is one grid unit in front of (x, y) in $g(x, y).inf[1]$. The information includes the path of the route and the locations of the buffers. Similarly, the other best routes such that the previous buffer is inserted into a grid which is k grid units in front of the grid (x, y) can be stored in $g(x, y).inf[k]$. Remember that k is upper bounded by up.

First, we initialize $g(x_0, y_0).inf[i]$ for $0 \leq i \leq up$. Then we will find

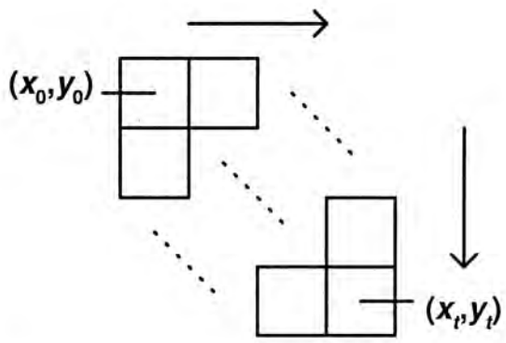


Figure 6.5: Procedures of finding routes

the routes dynamically from left to right and from top to bottom as shown in figure 6.5. When we want to find the best routes passing from (x_0, y_0) to $(x_0 + 1, y_0 + 1)$, the routes $l_{(x_0,y_0+1,1)}$, $l_{(x_0,y_0+1,2)}$, ..., $l_{(x_0,y_0+1,up)}$ and the routes $l_{(x_0+1,y_0,1)}$, $l_{(x_0+1,y_0,2)}$, ..., $l_{(x_0+1,y_0,up)}$ have already been found where $l_{(x,y,i)}$ the best route passing from (x_0, y_0) to (x, y) such that the previous buffer is inserted into a grid that is i grid units in front of (x, y). If we want to find the routes passing from (x_0, y_0) to $(x_0 + 1, y_0 + 1)$ such that the previous buffer is inserted into a grid that is $i + 1$ grid units in front of $(x_0 + 1, y_0 + 1)$, we can simply compare the routes $l_{(x_0+1,y_0,i)}$ and $l_{(x_0,y_0+1,i)}$. If $l_{(x_0+1,y_0,i)}$ is a better route based on our criteria, the route $l_{(x_0+1,y_0+1,i+1)}$ will pass through the grid $(x_0 + 1, y_0)$ and the information of the route will be stored in $g(x_0 + 1, y_0 + 1).inf[i + 1]$. In addition, we will find the best route among $l_{(x_0+1,y_0+1,low)}$, ..., $l_{(x_0+1,y_0+1,up)}$ and the information of the route will be stored in $g(x_0 + 1, y_0 + 1).inf[0]$.

Finally, we can find out the routes $l_{(x_t,y_t,1)}$, $l_{(x_t,y_t,2)}$, ..., $l_{(x_t,y_t,up)}$ dynamically and the information of the best route from (x_0, y_0) to (x_t, y_t) will be stored in $g(x_t, y_t).inf[0]$. The information includes the path of the route and the locations of the buffers.

Consider an example from figure 6.2, we can follow the algorithm in figure 6.4 to obtain the same best route. The intermediate values are shown in

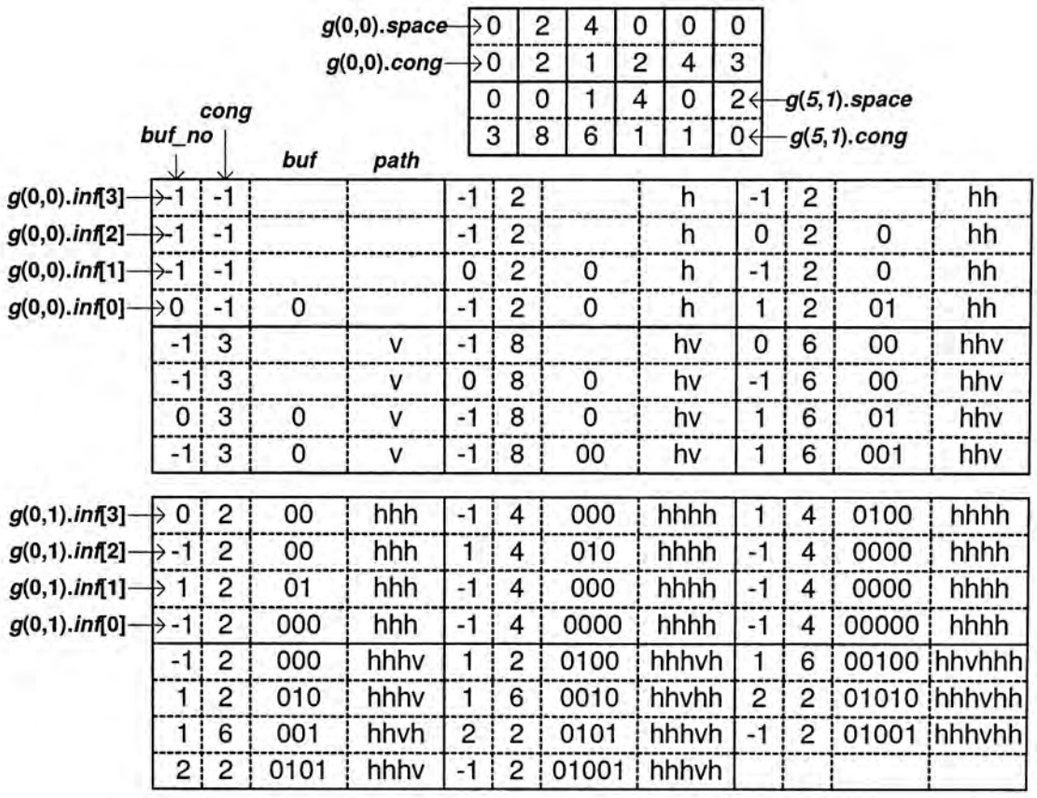
figure 6.6. First, we initialize $g(0,0).inf[i]$ for $0 \leq i \leq up$ by the following equations.

$g(0,0).inf[i].cong$ $(0 \leq i \leq up)$	$= -1$	(maximum congestion along the route)
$g(0,0).inf[i].path$ $(0 \leq i \leq up)$	$= NULL$	(path of the route)
$g(0,0).inf[i].buf$ $(0 \leq i \leq up)$	$= NULL$	(location of buffers)
$g(0,0).inf[i].buf_no$ $(1 \leq i \leq up)$	$= -1$	(number of buffer insertions)
$g(0,0).inf[0].buf_no$	$= 0$	

Remember that $g(x,y).inf[i]$ is the information of the routes passing from $(0,0)$ to (x,y) such that the previous buffer is inserted into a grid that is i grid units in front of the grid (x,y).

At the grid $(1,1)$, we will select the better route among $l_{(0,1,i)}$ and $l_{(1,0,i)}$ for $0 \leq i \leq 2$. Since $g(0,1).inf[i].cong$ are greater than $g(1,0).inf[i].cong$ for $0 \leq i \leq 2$ that means the routes passing through the grid $(0,1)$ are more congested than the routes passing through the grid $(1,0)$, $l_{(1,0,i)}$ is selected. As $g(1,1).cong$ is eight which is greater than $g(1,0).inf[i].cong$, $g(1,1).inf[i+1].cong$ become eight for $0 \leq i \leq 2$. It is because we need to update the maximum congestion on the routes. In addition, it is impossible to insert a buffer at the grid $(1,1)$, so $g(1,1).inf[0].buf_no$ is still -1. The value of $g(1,1).inf[0].buf$ becomes "00" where a '1' ('0') at the nth position in the sequence means that a buffer (no buffer) is inerted at the grid which is n grid units from $(0,0)$ along the route. In addition, $g(1,1).inf[0].path$ is "hv" since the route from the grid $(1,0)$ is selected and $(1,0)$ is above $(1,1)$ (denoted by 'v').

At the grid $(3,1)$, we will select the better route among $l_{(3,0,i)}$ and $l_{(2,1,i)}$ for $0 \leq i \leq 2$. Since $l_{(3,0,1)}$ and $l_{(3,0,2)}$ is better than $l_{(2,1,1)}$ and $l_{(2,1,2)}$ that means the routes passing through the grid $(3,0)$ is less congested, $g(3,1).inf[i+1]$ is equal to $g(3,0).inf[i]$ for $1 \leq i \leq 2$. However, buffer cannot be inserted at the grid $(3,0)$ and buffer can be inserted at the grid $(2,1)$, so $l_{(2,1,0)}$ is better



0	2	4	0	0	0						
0	2	1	2	4	3						
0	0	1	4	0	2						
3	8	6	1	1	0						

-1	-1			-1	2		h	-1	2		hh
-1	-1			-1	2		h	0	2	0	hh
-1	-1			0	2	0	h	-1	2	0	hh
0	-1	0		-1	2	0	h	1	2	01	hh
-1	3		v	-1	8		hv	0	6	00	hhv
-1	3		v	0	8	0	hv	-1	6	00	hhv
0	3	0	v	-1	8	0	hv	1	6	01	hhv
-1	3	0	v	-1	8	00	hv	1	6	001	hhv

0	2	00	hhh	-1	4	000	hhhh	1	4	0100	hhhh
-1	2	00	hhh	1	4	010	hhhh	-1	4	0000	hhhh
1	2	01	hhh	-1	4	000	hhhh	-1	4	0000	hhhh
-1	2	000	hhh	-1	4	0000	hhhh	-1	4	00000	hhhh
-1	2	000	hhhv	1	2	0100	hhhvh	1	6	00100	hhvhhh
1	2	010	hhhv	1	6	0010	hhvhh	2	2	01010	hhhvhh
1	6	001	hhvh	2	2	0101	hhhvh	-1	2	01001	hhhvhh
2	2	0101	hhhv	-1	2	01001	hhhvh				

Figure 6.6: Example of the algorithm

than $l_{(3,0,0)}$ and $g(3,1).inf[1]$ is equal to $g(3,0).inf[0]$. Finally, if we want to insert a buffer at the grid $(3,1)$, there is one possible route only, which is the route $l_{(3,1,2)}$. As a result, $g(3,1).inf[0]$ is equal to $g(3,1).inf[2]$. Then $g(3,1).inf[0].buf_no$ is increased by 1 and $g(3,1).inf[0].buf$ becomes "0101". In addition, $g(3,1).inf[0].path$ is "hhhv" as the route from the grid $(3,0)$ is selected and $(3,0)$ is above $(3,1)$.

Similarly, we can obtain $g(5,1).inf[i]$ for $1 \leq i \leq 3$ dynamically. Then we find two possible routes, one is "hhhvhh" that passes through the grids $(0,0)$, $(1,0)$, $(2,0)$, $(3,0)$, $(3,1)$, $(4,1)$ and $(5,1)$, the maximum congestion is 6 and one buffer insertion is required and the another one is "hhvhhh" that passes through the grids $(0,0)$, $(1,0)$, $(2,0)$, $(2,1)$, $(3,1)$, $(4,1)$ and $(5,1)$, the maximum congestion is 6 and two buffer insertions are required. Actually, they are route 1 and route 4 in figure 6.3 respectively. Finally, we will pick the

best route, route 4.

6.7 Summary

In this chapter, we have presented a simple global router based on our routing methodology and the router can perform routing efficiently by using dynamic programming. There are some limitations on this router. It will route each wire once only and all wires must be routed in their shortest Manhattan distances. However, this router is detailed enough to evaluate a floorplan output which is at very early stage of the design cycle.

Chapter 7

Interconnect-Driven Floorplanning by Alternative Packings

7.1 Introduction

Interconnect-driven floorplanners can perform congestion optimization and buffer planning effectively but there is a significant penalty on runtime. In this chapter, we propose an approach to reduce interconnect cost by searching alternative packings. We have found that if there is a packing F containing some rectangular supermodules, we can rearrange the blocks in the supermodules to obtain a new packing with the same area as F but with possibly smaller interconnect cost. In general, we can apply this approach on the final floorplan solution for different purposes such as congestion reduction and buffer planning.

7.2 Overview of the Method

In this section, we will have an overview of the method of searching alternative packings. We define an alternative packing of a floorplan as follows.

Definition 7.1 Given a floorplan F of a set of modules, an alternative packing of F is another floorplan that has the same area as F independent of the module dimensions.

An example of alternative packings is shown in figure 7.1. In our method, we want to find all the alternative packings from a floorplan solution. Generally, alternative packings can be obtained by flipping some rectangular supermodules horizontally, vertically or both horizontally and vertically.

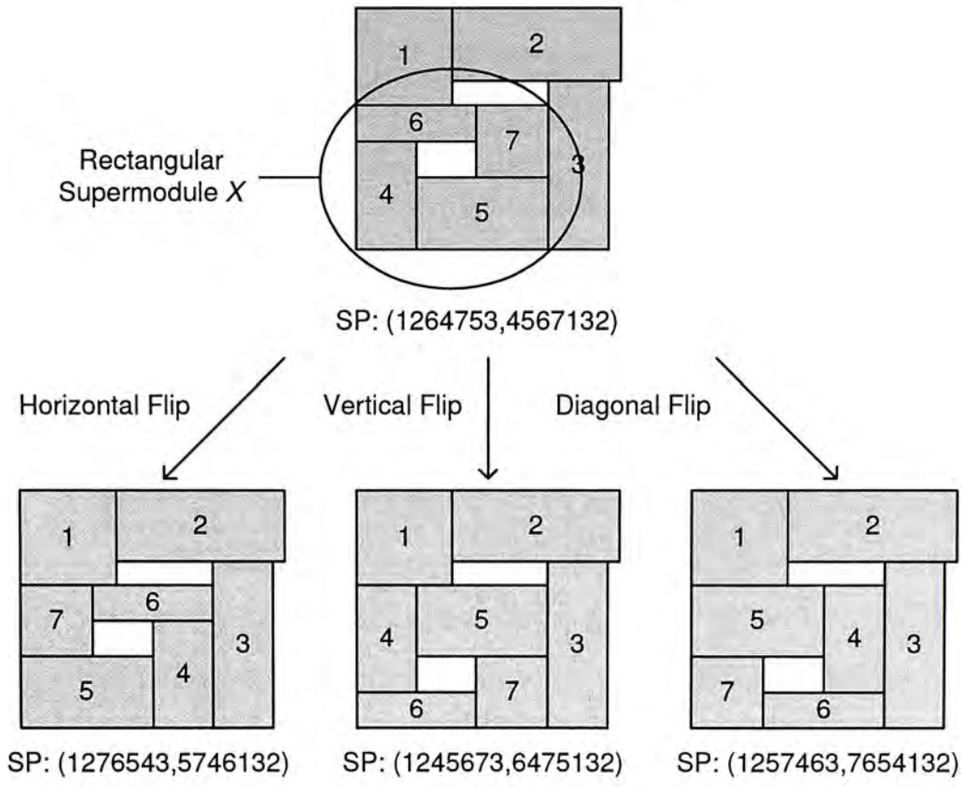


Figure 7.1: Example of alternative packings

Since all alternative packings have the same area, their areas are not required to be calculated again. However, the interconnect costs may be different because the locations of some modules have been changed. We can then find one with the minimum interconnect cost among all the alternative packings. It means that the interconnect cost can be further optimized while keeping the

area of the packing unchanged. In our floorplanner, we will apply this method to further optimize the interconnect cost of the final floorplan.

7.3 Searching Alternative Packings

In figure 7.1, there is a rectangular supermodule X containing four modules in the packing. We can change the packings while preserving the area by three ways: flipping X horizontally, flipping X vertically and flipping X horizontally and vertically (we call this a diagonal flip). In the method of searching alternative packings, we will use sequence pair to represent a floorplan. We can obtain alternative packings with the same area by working on the sequence pair representations only. In this section, we will discuss how we can find an alternative sequence pair from a given sequence pair (S_1, S_2) such that the packing represented by the alternative sequence pairs will have the same area as that represented by (S_1, S_2) independent of the dimensions of the modules.

7.3.1 Rectangular Supermodules in Sequence Pair

In order to construct alternative packings, we need to find all the rectangular supermodules in the original floorplan solution first. We define a *rearrangable module set* in a sequence pair as follows.

Definition 7.2 Given a sequence pair (S_1, S_2), a set of two or more modules form a rearrangable module set if they form contiguous sub-sequences in both S_1 and S_2.

An example is shown in figure 7.1. In this example, the packing is represented by the sequence pair $(1264753, 4567132)$. The set of modules $4, 5, 6, 7$ have formed contiguous sub-sequences in both S_1 and S_2. Thus, the set of modules $\{4, 5, 6, 7\}$ is a rearrangable module set according to definition 7.2. In

addition, the set of modules $\{4, 5, 6, 7\}$ will form a rectangular supermodule in the packing according to the following theorem.

Corollary 7.3 Given a sequence pair (S_1, S_2) and its corresponding packing F, the set of modules in a rearrangable module set will form a rectangular supermodule in F independent of the dimensions of the modules.

Proof Given a set of modules M_r in a rearrangable module set. The modules in M_r form a contiguous sub-sequence s_{r1} in S_1 and form a contiguous sub-sequence s_{r2} in S_2. Consider the relationships between the modules in M_r and the other modules not in M_r. If a module $m_i \notin M_r$ is on the left of a module in M_r, the sequence pair should be $(...m_i...s_{r1}..., ...m_i...s_{r2}...)$. It means that m_i is on the left of all the modules in M_r. The same argument follows for m_i lying on the right of the modules in M_r. Similarly, if a module $m_i \notin M_r$ is above a module in M_r, the sequence pair should be $(...m_i...s_{r1}..., ...s_{r2}...m_i...)$. It means that m_i is above all the modules in M_r. The same argument follows for m_i lying below the modules in M_r. Thus, the horizontal and vertically relationships between the modules in M_r and all the other modules not in M_r are identical. As a result, the set of modules in a rearrangable module set will form a rectangular supermodule in the packing independent of the dimensions of the modules. □

7.3.2 Finding rearrangable module sets

In order to construct all the alternative packings (alternative sequence pairs) of a given floorplan F, we need to find all the rectangular supermodules (rearrangable module sets) in F. In this section, we will discuss how we can find all rearrangable module sets from a given sequence pair effectively. Given a sequence pair (S_1, S_2) where $S_1 = s_{11}s_{12}...s_{1n}$ and $S_2 = s_{21}s_{22}...s_{2n}$ where n is the total number of modules. If a contiguous sub-sequence in S_1 contains two modules, the sub-sequence should be $(s_{1i}s_{1i+1})$ for some $1 \leq i \leq n-1$.

If a contiguous sub-sequence in S_1 contains three modules, the sub-sequence should be $(s_{1i}s_{1i+1}s_{1i+2})$ for some $1 \leq i \leq n-2$. Similarly, if a contiguous sub-sequence in S_1 contains $n-1$ modules, the sub-sequence should be $(s_{11}...s_{1n-1})$ or $(s_{12}...s_{1n})$. Notice that the sub-sequences cannot contain more than $n-1$ modules. The total number of contiguous sub-sequences in S_1:

$$\begin{aligned} & 2 + ... + (n-2) + (n-1) \\ = \ & \tfrac{(n+1)(n-2)}{2} \end{aligned}$$

By definition 7.2, a rearrangable module set in the sequence pair (S_1, S_2) should form contiguous sub-sequences in both S_1 and S_2. Thus, the maximum number of rearrangable module sets by looking at S_2 is also equal to $\frac{(n+1)(n-2)}{2}$. In order to find all the rearrangable module sets, we can scan all the possible contiguous sub-sequences from S_1 or S_2 and check whether they also form a contiguous sub-sequence in the other sequence. We propose the algorithm FRSS (Find rearrangable module set) to find all the rearrangable module sets. The FRSS algorithm is shown in figure 7.2.

In the algorithm, we try to find all the rearrangable module sets by sequential search. First we obtain a sub-sequence s in S_1. Then we can find the first and the last position in S_2 for those modules in s. If the difference between the first and the last position is equal to the number of modules in s, s is a rearrangable module set. In this algorithm, we will first scan the sub-sequence $s_{11}s_{12}$, then we will scan the sub-sequences $s_{12}s_{13}$ and $s_{11}s_{12}s_{13}$. Finally, we will scan the sub-sequences $s_{1n-1}s_{1n}$, $s_{1n-2}s_{1n-1}s_{1n}$, ..., $s_{11}...s_{1n}$. Once we find a rearrangable module set, we will store the starting and ending position of the rearrangable module set in S_2 in the array $rs[]$. An example is shown in figure 7.3.

Consider the example in figure 7.3, we have the packing represented by the sequence pair (S_1, S_2) where $S_1 = abcdefg$ and $S_2 = fcbdgae$. The x-axis i is the last position of the sub-sequence in S_1 and the y-axis j is the first position of the sub-sequence in S_1.

FRSS Algorithm
Input: Total number of modules, n
A sequence pair, $(s_{11}...s_{1n}, s_{21}...s_{2n})$
Output: Total number of rearrangable module sets, *count*
An array of the rearrangable module sets, $rs[]$
$count = 0$
for each j where $1 \leq j \leq n$
  $ssp[j].low = s_{1j}.pos$
  $ssp[j].up = s_{1j}.pos$ ($s_{1j}.pos$ is the position of s_{1j} in S_2)
for $i = 2$ to n
  for $j = i - 1$ downto 1
    if $ssp[j].low > s_{1i}.pos$
      $ssp[j].low = s_{1i}.pos$
    if $ssp[j].up < s_{1i}.pos$
      $ssp[j].up = s_{1i}.pos$
    if $(ssp[j].up - ssp[j].low) = (i - j)$ and $(0 < i - j < n)$
      $rs[count].end = ssp[j].up$
      $rs[count].start = ssp[j].low$
      $count = count + 1$

Figure 7.2: The algorithm of FRSS

At the beginning, $ssp[j].low$ and $ssp[j].up$ are all initialized to $s_{1j}.pos$ where $s_{1j}.pos$ is the position of the module s_{1j} in S_2. When we consider $i = 2$ and $j = 1$, this corresponds to the sub-sequence ab. The position of a and b in S_2 are 3 and 6 respectively, so $ssp[1].low$ is 3 and $ssp[1].up$ is 6. Then $ssp[1].up - ssp[1].low + 1$ is equal to 4. It means that the shortest sub-sequence in S_2 containing both a and b has four modules. However, there are only two modules in the sub-sequence ab in S_1 (it can be computed by $i - j + 1$). It means that ab cannot form a contiguous sub-sequence in S_2, so it is not a rearrangable module set.

When we consider $i = 3$ and $j = 2$, this corresponds to the sub-sequence bc. The position of b and c in S_2 are 2 and 3 respectively, so $ssp[2].low$ is 2 and $ssp[2].up$ is 3. $ssp[2].up - ssp[2].low + 1$ is equal to 2, so the shortest sub-sequence in S_2 containing both b and c has two modules. Since there are

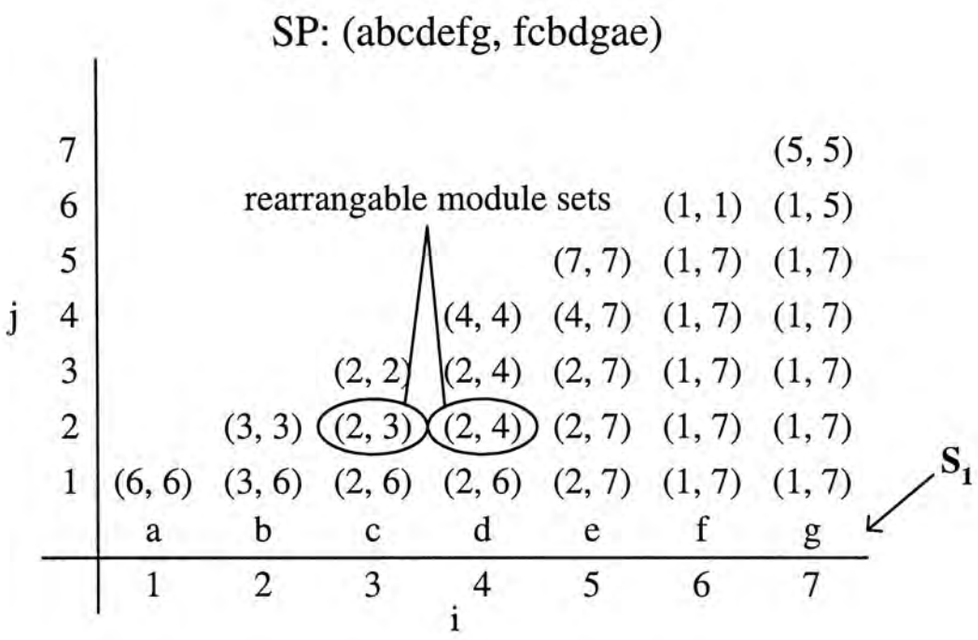


Figure 7.3: An example of finding rearrangable module sets by FRSS

also two modules in the sub-sequence *bc*, *bc* forms contiguous sub-sequences in both S_1 and S_2 and it is a rearrangable module set.

Similarly, when we consider $i = 4$ and $j = 2$, this corresponds to the sub-sequence *bcd*. The position of b, c and d in S_2 are 2, 3 and 4 respectively, so $ssp[2].low$ is 2 and $ssp[2].up$ is 4. $ssp[2].up - ssp[2].low + 1$ is equal to 3. It means that the shortest sub-sequence in S_2 containing b, c and d has three modules. Since there are three modules in the sub-sequence *bcd*, *bcd* forms contiguous sub-sequences in both S_1 and S_2 and it is a rearrangable module set. By this algorithm, we will scan all the possible sub-sequences and we have the following theorem.

Corollary 7.4 Given a sequence pair (S_1, S_2) of a packing F, all the rearrangable module sets in (S_1, S_2) can be found by the algorithm FRSS.

7.3.3 Alternative Sequence Pairs

In order to construct alternative packings of a given floorplan F described by a sequence pair (S_1, S_2), we need to find the corresponding alternative sequence pairs which are obtained by rearranging the modules in one or more rearrangable module sets in (S_1, S_2). According to theorem 7.3, the modules in a rearrangable module set should form a rectangular supermodule in the packing. The rearrangements should be identical to performing horizontal flips, vertical flips or diagonal flips on the supermodule.

Given a sequence s, we use $\tilde{s}$ to denote the sequence obtained by writing s in the reversed order. For example, if $s = 123$, $\tilde{s} = 321$. We can perform horizontal and vertical flip by swapping or reversing the rearrangable module sets. An example is shown in figure 7.4.

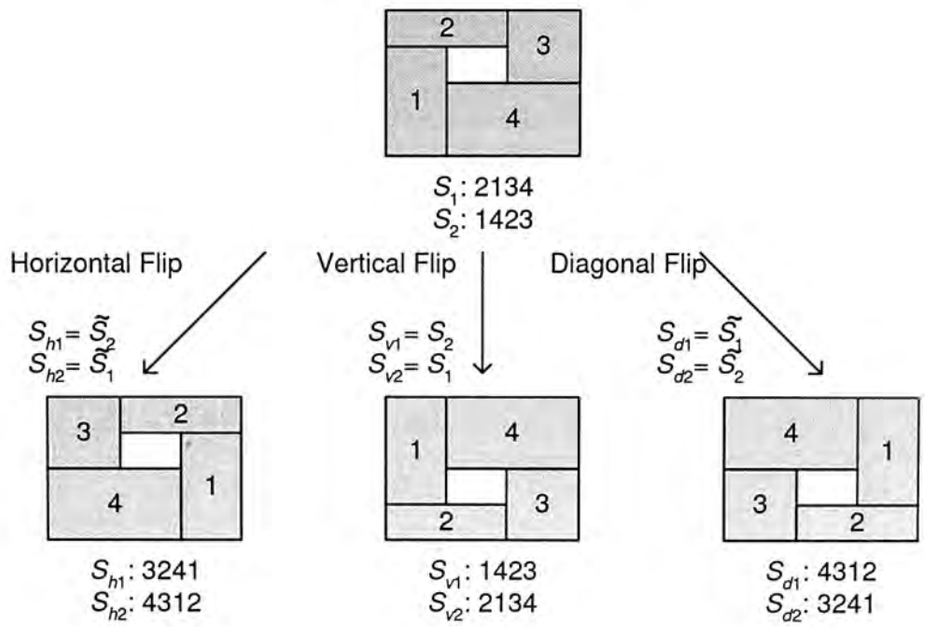


Figure 7.4: Example of alternative sequence pairs

Vertical Flip

Figure 7.4 shows a packing corresponding to the sequence pair $(S_1, S_2) = (2134, 1423)$. Note that any packing is a rectangular supermodule by itself. When we perform a vertical flip, we swap S_1 and S_2: $S_{v1} = S_2$ and $S_{v2} = S_1$. Before swapping, if a module m_i is on the left of m_j in the initial packing, m_i should be in front of m_j in both S_1 and S_2. After swapping, m_i is still in front of m_j in both S_{v1} and S_{v2}. Thus, the horizontal relationships between the modules are preserved. On the other hand, if a module m_i is above m_j before swapping, m_i should be in front of m_j in S_1 and after m_j in S_2. After swapping, m_i becomes after m_j in S_{v1} and in front of m_j in S_{v2}. Thus, the vertical relationships between the modules will be reversed. An illustration is shown in figure 7.5.

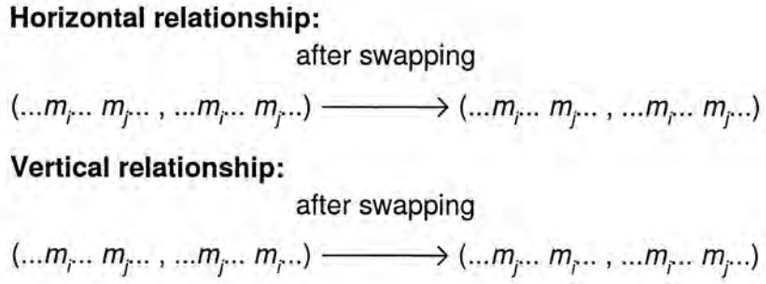


Figure 7.5: Vertical flip

Horizontal Flip

When we perform horizontal flip, we reverse and swap the sequences: $S_{h1} = \tilde{S}_2$ and $S_{h2} = \tilde{S}_1$. If a module m_i is on the left of m_j in the original packing, m_i should be in front of m_j in both S_1 and S_2 before the reversal but m_i will become after m_j after the reversal. Then we perform swapping which has no effect on the horizontal relationships between the modules. As a result, the horizontal relationships between the modules will be reversed. On the other

hand, if a module m_i is above m_j in the original packing, m_i should be in front of m_j in S_1 and after m_j in S_2. After the reversal, m_i will become after m_j in $\tilde{S_1}$ and in front of m_j in $\tilde{S_2}$. Then we perform swapping by which the vertical relationships will be reversed again. As a result, the vertical relationships between the modules are unchanged. An illustration is shown in figure 7.6.

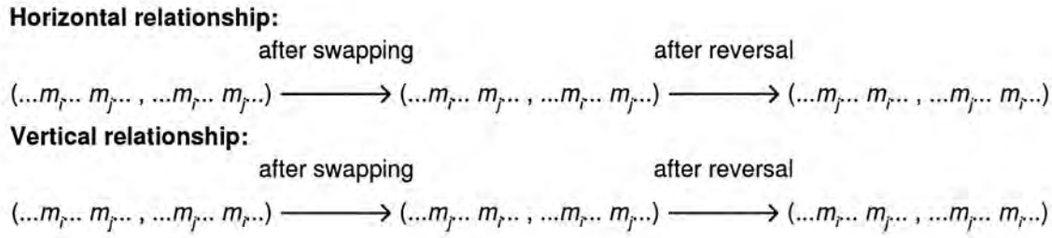


Figure 7.6: Horizontal flip

Diagonal Flip

Finally, we perform diagonal flip. Actually, this can be considered as performing a horizontal or vertical flip first and then followed by the other one. Thus $S_{d1} = S_{h2} = \tilde{S_1}$ and $S_{d2} = S_{h1} = \tilde{S_2}$, or $S_{d1} = \tilde{S_{v2}} = \tilde{S_1}$ and $S_{d2} = \tilde{S_{v1}} = \tilde{S_2}$. As a result, $S_{d1} = \tilde{S_1}$ and $S_{d2} = \tilde{S_2}$. An illustration is shown in figure 7.7.

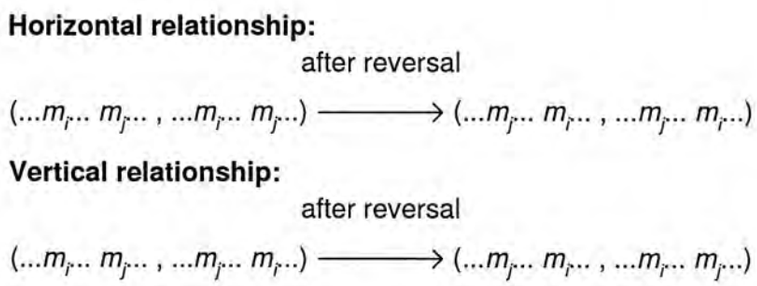


Figure 7.7: Diagonal flip

After finding all the rearrangable module sets, we can obtain the alternative packings by applying vertical flips, horizontal flips or diagonal flips on those

sub-sequences. As we can apply three types of operations on each rearrangable module set, the total number of alternative packings of a floorplan F is upper bounded by 4^k where k is the number of rearrangable module sets in the sequence pair representation of F. It may not be exactly equal to 4^k because some operations may result in the same packing. An example is shown in figure 7.8. From the example, the packing F is identical to the packing F_v. F_v is obtained by flipping the rectangular supermodule Y in F vertically but F is identical to F_v because the modules in Y are horizontally related only.

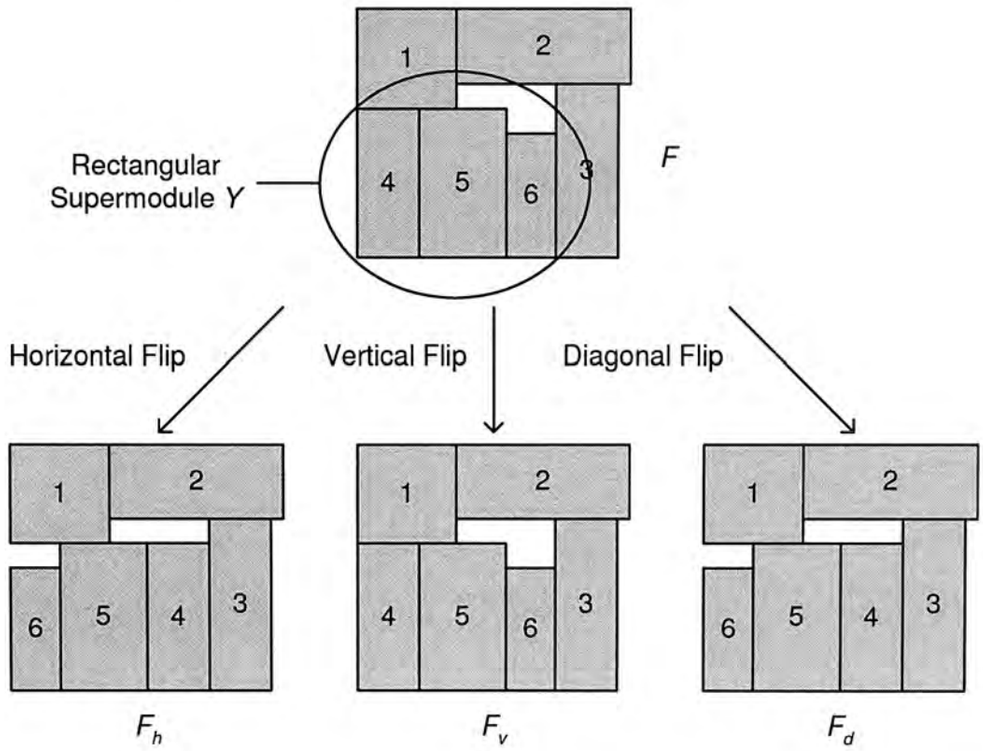


Figure 7.8: Example of identical alternative packing

7.4 Implementation

In order to evaluate the method of searching alternative packings on improving interconnect cost, we have implemented this method in an interconnect-driven

floorplanner that uses half-perimeter estimation on wirelength calculation and uses sequence pair as the floorplan representation. In this floorplanner, we propose a three step process to reduce interconnect cost. First, we find all the rearrangable module sets of a give sequence pair (S_1, S_2) by FRSS. Then the alternative sequence pairs are obtained by swapping and reversing some rearrangable module sets in (S_1, S_2). Finally, we optimize the interconnect cost by selecting the alternative packing with the minimum wirelength. Since all the alternative packings have the same area, there is no area penalty on picking different alternative packings.

7.4.1 Re-calculation of Interconnect Cost

After we find the alternative sequence pairs, we need to re-calculate the interconnect cost. It is time consuming if we need to re-construct the horizontal and vertical constraint graphs whenever an alternative sequence pair is obtained.

In our floorplanner, we can calculate the new positions of the modules in the alternative packings by the following method. First of all, a rearrangable module set will form a rectangular supermodule in the packing. We can thus obtain the co-ordinates of the upper right corner (x_{up}, y_{up}) and the co-ordinates of the lower left corner (x_{low}, y_{low}) of the rectangular supermodule. Then we can estimate the new positions of the modules in the rearrangable module sets by the following equations according to the operations. An example is shown in figure 7.9.

For horizontal flip,

$$\begin{aligned} x_{new} &= x_{low} + (x_{up} - x_{old} - length) \\ y_{new} &= y_{old} \end{aligned} \tag{7.1}$$

For vertical flip,

$$\begin{aligned} x_{new} &= x_{old} \\ y_{new} &= y_{low} + (y_{up} - y_{old} - height) \end{aligned} \tag{7.2}$$

For diagonal flip,

$$\begin{aligned} x_{new} &= x_{low} + (x_{up} - x_{old} - length) \\ y_{new} &= y_{low} + (y_{up} - y_{old} - height) \end{aligned} \tag{7.3}$$

where (x_{old}, y_{old}) is the position of the module before flipping, (x_{new}, y_{new}) is the position of the module after flipping, and *length* and *height* are the length and height of the module respectively.

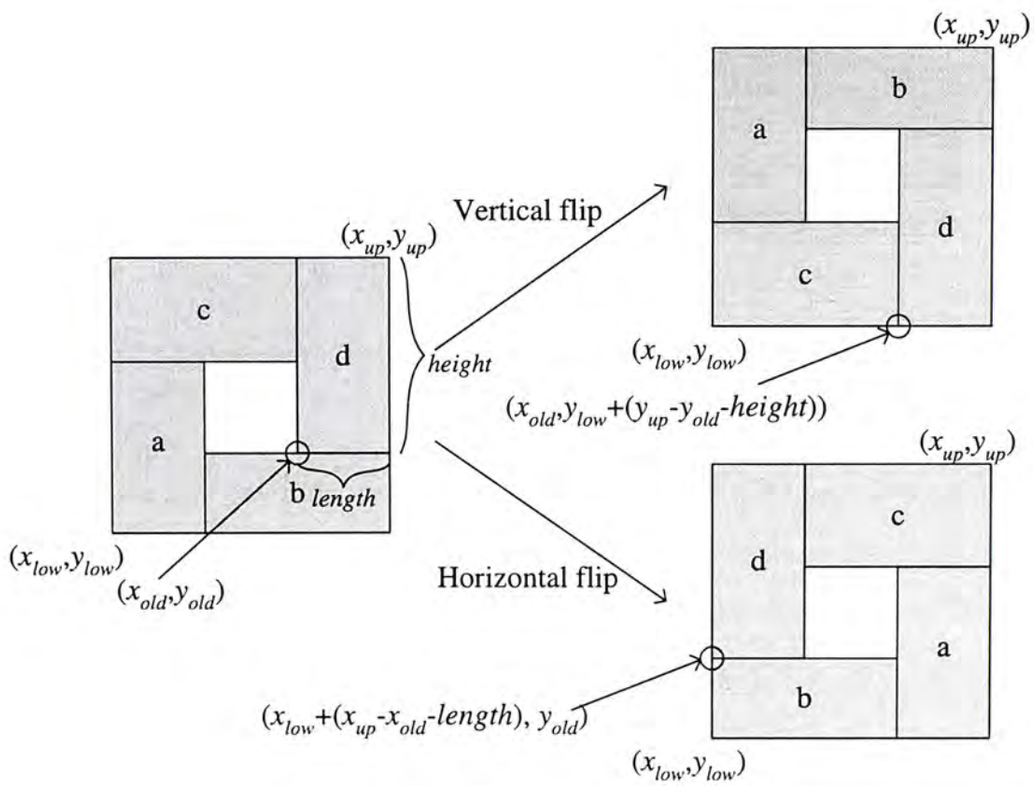


Figure 7.9: Re-calculation of the positions of the modules in the new packing

From the above equations, we can find the new positions of the modules in the alternative packings efficiently. It is because re-construction of the

vertical and horizontal constraint graphs is not required. However, the actual positions will be slightly different from the estimated ones. An example is shown in figure 7.10.

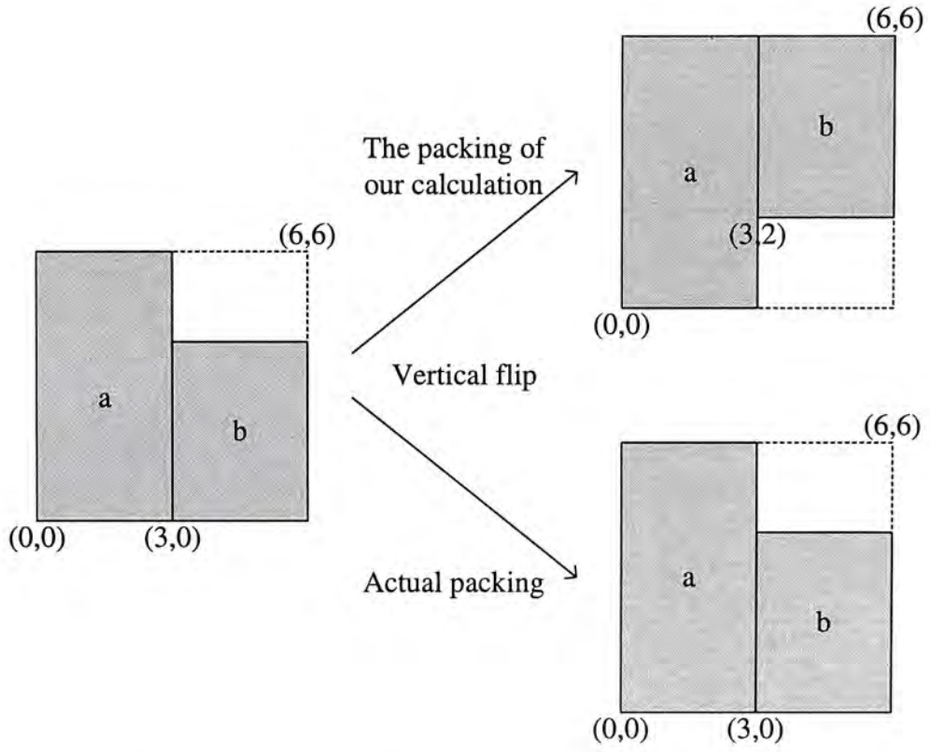


Figure 7.10: Difference between the actual positions of the module and the estimated positions

Consider the example in figure 7.10, if we perform a vertical flip, the positions of the modules should not be changed. However, the positions of module b will be changed from (3,0) to (3,2) according to the equation 7.2. However, it does not influence the area of the packing. In addition, the difference in wirelength is very small when half-perimeter is employed as the estimation method. Results show that the difference should be less than 0.05%.

7.4.2 Cost Function

We use simulated annealing in our floorplanner. Simulated annealing is an iterative and non-deterministic optimization technique. We use the following cost function:

$$Cost = Area + \alpha \times Wire$$

where $Area$ is the area of the floorplan, $Wire$ is the total wirelength (half-perimeter estimation is used), α is a parameter. This parameter will be set at the beginning of simulated annealing process according to the ratio of importance of the area term and the wirelength term. This can be done by performing a sequence of random walks at the beginning of the annealing process and sampling the average values of these penalty terms. The value of α can then be computed accordingly.

7.4.3 Time Complexity

According to the algorithm of FRSS, we need to scan all the possible sub-sequences. If the packing contains n modules, there will be 2 sub-sequences with $n-1$ modules, 3 sub-sequences with $n-2$ modules, ..., and n sub-sequences with one module. Therefore, we need to scan $2+3+...+n-1+n$ times which is equal to $\frac{(n+2)(n-1)}{2}$. As a result, the time complexity is $O(n^2)$.

7.5 Experimental Results

We have implemented two floorplanners, a floorplanner $F1$ based on simulated annealing that applies the three step process of searching alternative packings to reduce the interconnect cost of the final floorplan solution only and a floorplanner $F2$ that applies the three step process to reduce the interconnect cost of every intermediate floorplan solution in the annealing process.

In the experiments, we use half-perimeter estimation in wirelength computation. The data sets used in the experiments are *hp*, *ami*33, *ami*49, *playout* and six randomly generated data sets. Each result shown in table 7.2 and 7.3 is the average value obtained by performing an experiment eight times. The information of the data sets are shown in table 7.1.

Cases	No. of modules	No. of nets
*rand*30_1	30	1000
*rand*30_2	30	1000
*rand*30_3	30	1000
*rand*50_1	50	2500
*rand*50_2	50	2500
*rand*50_3	50	2500
hp	10	83
*ami*33	33	123
*ami*49	49	408
playout	62	1161

Table 7.1: Information of data sets

Table 7.2 shows the improvements in wirelength after searching alternative packings of the final floorplan solution. As the time penalty for searching alternative packings of the final floorplan solution is so small, i.e., less than 0.1% for all test cases, and the improvement on wirelength is guaranteed, it is desirable to apply the method. However, the average improvement on wirelength is not big because the number of alternative packings is limited and the final solution is already optimized in interconnect cost by the annealing process.

From table 7.2, we can see that the improvement will increase when the weight of wirelength in the cost function is reduced. It is because when the weight of wirelength in the cost function is reduced, the wirelength will be less optimized in the solution of the simulated annealing process. This may result in an increase in the differences in wirelength between a packing and its alternative packings. As a result, both the average improvement and the

Cases	[1]$r = 2$		$r = 4$	
	Mean Improvement in Wirelength	Maximum Improvement in Wirelength	Mean Improvement in Wirelength	Maximum Improvement in Wirelength
*rand*30_1	0.11%	0.88%	0.32%	1.05%
*rand*30_2	0.06%	0.48%	0.07%	0.53%
*rand*30_3	0.13%	1.04%	0.32%	1.55%
*rand*50_1	0.12%	0.95%	0.23%	1.74%
*rand*50_2	0.03%	0.28%	0.12%	0.93%
*rand*50_3	0.08%	0.67%	0.46%	1.94%
hp	0.18%	1.28%	0.52%	4.25%
*ami*33	0.16%	0.95%	0.36%	2.74%
*ami*49	0.06%	0.39%	0.26%	2.01%
playout	0.06%	0.19%	0.07%	0.49%

[1] $\alpha \approxeq \frac{wirelength*r}{area}$

Table 7.2: Maximum and average improvement on wirelength by applying the method of searching alternative packings on the final solution floorplan

maximum improvement increase when the weight of wirelength in the cost function is reduced.

We compare the floorplanner $F1$ and $F2$ in table 7.3. We can see that the result between $F1$ and $F2$ is similar. However, the runtime of $F1$ is much faster than that of $F2$. We can conclude that applying the three step process of searching alternative packings to reduce the interconnect cost of the final floorplan solution is good enough and there is no need to apply the method to all intermediate floorplan solutions in the annealing process.

7.6 Summary

In this chapter, we present a new method to reduce interconnect cost by searching alternative packings. We found that if a packing F contains some rectangular supermodules, we can rearrange the blocks in the supermodules to obtain a

Cases		Area ($10^3 \mu m^2$)	Wirelength ($10^3 \mu m$)	[1] Wirelength ($10^3 \mu m$)	Runtime (s)
*rand*30_1	$F1$	561.64	656.83	652.56	73.98
	$F2$	562.30	655.49	—	381.79
*rand*30_2	$F1$	355.14	592.47	591.32	71.59
	$F2$	356.72	589.98	—	441.64
*rand*30_3	$F1$	279.36	525.12	521.08	73.18
	$F2$	280.42	522.08	—	539.19
*rand*50_1	$F1$	578.54	1824.25	1820.81	188.55
	$F2$	577.98	1815.46	—	708.63
*rand*50_2	$F1$	509.09	1786.42	1784.25	188.06
	$F2$	511.21	1783.15	—	612.93
*rand*50_3	$F1$	929.61	2392.52	2380.09	192.37
	$F2$	928.98	2382.12	—	639.12
hp	$F1$	9.881	5.399	5.389	1.69
	$F2$	9.879	5.340	—	7.56
*ami*33	$F1$	1213.47	73.55	73.25	13.49
	$F2$	1213.30	73.49	—	81.79
*ami*49	$F1$	38080	1093.67	1092.09	33.18
	$F2$	37940	1117.91	—	89.09
playout	$F1$	957.75	676.26	675.81	101.36
	$F2$	955.14	680.63	—	308.56

[1] Further optimization by searching alternative packings.

Table 7.3: Comparisons between $F1$ and $F2$

new packing with the same area as F but possibly with an improved interconnect cost. In our implementation, we find all the rearrangable module sets in a sequence pair and then obtain the alternative sequence pairs by performing swapping or reversing on the rearrangable module sets. The wirelengths of the alternative packings can be re-calculated easily without re-construction of the horizontal and vertical constraint graphs. According to the experimental results, our floorplanner can reduce interconnect cost without any penalty in area or runtime.

Chapter 8

Conclusion

In this thesis, we have introduced the basic idea of floorplannning and have reviewed the related papers on interconnect-driven floorplanning. There are several previous works addressing the interconnect issues in floorplan design. Most of them will focus either on the congestion optimization or buffer planning.

Thus, we proposed a congestion model for estimating interconnect cost with buffer planning in a routability driven floorplanner. By using this probabilistic model, the number of blocked nets, delay and congestion can be estimated, taken into account the exact buffer locations and routing blockages. Experimental results show that congestion and delay can be better optimized using this congestion model with only a little penalty in area.

Instead of computing the exact locations of the buffers, we proposed a new congestion model to compute the congestion information under the variable interval buffer insertion constraint. The estimations are based on the supply and demand analysis of routing and buffer resources, where the supply is determined by the floorplan solution (the amount of empty space) and the demand is determined by the interconnect structure. Computations of the congestion information in our model are quite complicate but they can be performed efficiently by dynamic programming. Experimental results show that our floorplanner can reduce interconnect cost for all data sets.

Interconnect-driven floorplanners can perform congestion optimization and buffer planning effectively but there is also a significant penalty on runtime. Thus, we propose a method to reduce interconnect cost by searching alternative packings. We have found that if there is a packing F containing some rectangular supermodules, we can rearrange those blocks in the supermodules to obtain a new packing with the same area as F but have smaller interconnect cost. In general, the alternative packings can be obtained by flipping either horizontally or vertically or both horizontally and vertically on the rectangular supermodules. Experiments show that our floorplanner can reduce interconnect cost without any penalty in area and runtime.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flow. In *Prentice Hall*, 1993.

[2] C. J. Alpert, J. Hu, S. S. Sapatnekar, and P. G. Villarrubia. A practical methodology for early buffer and wire resource allocation. In *Proceedings of the 38th ACM/IEEE Design Automation Conference*, pages 189–194, 2001.

[3] C. C. Chang, J. Cong, D. Z. Pan, and X. Yuan. Interconnect-driven floorplanning with fast global wiring planning and optimization. In *Proc. SRC Tech. Conference*, 2000.

[4] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu. B*-trees: A new representation for non-slicing floorplans. In *Proceedings of the 37th ACM/IEEE Design Automation Conference*, pages 458–463, 2000.

[5] H. M. Chen, H. Zhou, F. Y. Young, D. Wong, H. H. Yang, and N. Sherwani. Integrated floorplanning and interconnect planning. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 354–357, 1999.

[6] J. Cong. Challenges and opportunities for design innovations in nanometer technologies. In *Frontiers in Semiconductor Research: A Collection of SRC Working Papers*, 1997.

[7] J. Cong, T. Kong, and D.Z. Pan. Buffer block planning for interconnect driven floorplanning. In *Digest of Technical Papers, IEEE Internation Cnference on Computer-Aided Design*, pages 358–363, 1999.

[8] T. Cormen, C. Leiserson, and R. Rivest. Introduction to algorithms. In *The MIT Press, Cambridge, Massachusetts 02142*, 1990.

[9] F. F. Dragan, A. B. Kahng, S. Muddu, and A. Zelikovsky. Provably good global buffering using an available buffer block plan. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 104–109, 2000.

[10] P. N. Guo, C. K. Cheng, and T. Yoshimura. An o-tree representation of non-slicing floorplan and its application. In *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pages 268–273, 1999.

[11] J. M. Ho, G. Vijayan, and C. K. Wong. A new approach to the rectilinear steiner tree problem. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 161–166, 1989.

[12] X. L. Hong, G. Huang, Y. Cai, and J. Gu. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 8–12, 2000.

[13] G. Huang, X. L. Hong, C. Qiao, and Y. Cai. A timing-driven block placer based on sequence pair model. In *Proceedings of ASP-ACM/IEEE Design Automation Conference*, pages 249–252, 1999.

[14] A. B. Kahng, S. Muddu, E. Sarto, and R. Sharma. Interconnect tuning strategies for high performance ics. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 471–478, 1998.

[15] M. Z. Kang and W. W. M. Dai. Arbitrary rectilinear block packing based on sequence pair. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 259–266, 1998.

[16] K. Kozminski and E. Kinnen. Rectangular dual of a planar graph for use in area planning for vlsi integrated circuits. In *Proceedings of 21st ACM/IEEE Design Automation Conference*, pages 655–656, 1984.

[17] J. Lillis, C.K. Cheng, and T. T. Y. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 138–143, 1996.

[18] C. Lin. A more efficient sequence pair perturbation scheme. In *IEEE/ProRISC99*, pages 295–300, 1999.

[19] C. Lin and D.M.W. Leenaerts. A new faster sequence pair algorithm. In *Proceedings of IEEE Internation Symposium in Circuits and Systems*, pages 407–410, 2000.

[20] J. Lou, S. Krishnamoorthy, and H. S. Sheng. Estimating routing congestion using probabilistic analysis. In *Proceedings of Internation Symposium on Physical Design*, pages 112–117, 2001.

[21] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-packing- based module placement. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 472–479, 1995.

[22] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. In *IEEE Transactions on CAD of Integrated Circuit and System*, pages 1518–1524, December 1996.

[23] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module placement on bsg-structure and ic layout applications. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 484–491, 1996.

[24] S. Nakatake, M. Furuya, and Y. Kajitani. Module placement on bsg-structure with pre-placed modules and rectilinear modules. In *Proceedings of ASP-ACM/IEEE Design Automation Conference*, pages 571–576, 1998.

[25] V. Narayananan, D. LaPotin, R. Gupta, and G. Vijayana. Pepper- a timing driven early floorplanner. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 230–235, 1995.

[26] T. Okamoto and J. Cong. Buffered steiner tree construction with wire sizing for interconnect layout optimization. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 44–49, 1996.

[27] R. H. J. M. Otten. Automatic floorplan design. In *Proceedings of 19th ACM/IEEE Design Automation Conference*, pages 261–267, 1982.

[28] R. H. J. M. Otten. Layout structure. In *Proceedings of IEEE Large Scale Systems Symposium*, 1982.

[29] R. H. J. M. Otten. Efficient floorplan optimization. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 499–502, 1983.

[30] Y. X. Pang, F. Balasa, K. Lampaert, and C. K. Cheng. Block placement with symmetry constraints based on the o-tree non-silicing representation. In *Proceedings of 37th ACM/IEEE Design Automation Conference*, pages 464–467, 2000.

[31] S. M. Sait, H. Youssef, S. Tanvir, and M. S. T. Benten. Timing influenced general-cell genetic floorplanner. In *Proceedings of the ASP-ACM/IEEE*

Design Automation Conference 95/CHDL 95 /VLSI 95, pages 135–140, 1995.

[32] K. Sakanushi, S. Nakatake, and Y. Kajitani. The multi-bsg: Stochastic approach to an optimum packing of convex-rectilinear blocks. In *Proceedings of IEEE Internation Cnference on Computer-Aided Design*, pages 267–274, 1998.

[33] P. Sarkar, V. Sundararaman, and C. K. Koh. Routability-driven repeater block planning for interconnect-centric floorplanning. In *Proceedings of Internation Symposium on Physical Design*, pages 186–191, 2000.

[34] C. W. Sham, W. C. Wong, and E. F. Y. Young. Congestion estimation with buffer planning in floorplan design. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 696–701, 2002.

[35] C. W. Sham and E. F. Y. Young. Routability-driven floorplanning with buffer planning. In *Proceedings of Internation Symposium on Physical Design*, pages 50–55, 2002.

[36] S. Sutanthavibul, E. Shragowitz, and J. Rosen. An analytical approach to floorplan design and optimization. In *IEEE Transactions on CAD of Integrated Circuit and System*, pages 761–769, June 1991.

[37] X. P. Tang, R. Q. Tian, and D.F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 106–111, 2000.

[38] X. P. Tang and D.F. Wong. Planning buffer locations by network flows. In *Proceedings of Internation Symposium on Physical Design*, pages 186–191, 2000.

[39] G. Vijayan and R. Tsay. A new method for floorplanning using topological constraint reduction. In *IEEE Transactions on CAD of Integrated Circuit and System*, pages 1494–1501, December 1991.

[40] M. Wang and M. Sarrafzadeh. On the behavior of congestion minimization during placement. In *Proceedings of Internation Symposium on Physical Design*, pages 145–150, 1999.

[41] M. Wang and M. Sarrafzadeh. Modeling and minimization of routing congestion. In *Proceedings of the ASP-ACM/IEEE Design Automation Conference*, pages 185–190, 2000.

[42] M. Wang, X. Yang, and M. Sarrafzadeh. Congestion minimization during placement. In *IEEE Transactions on CAD of Integrated Circuit and System*, pages 1140–1148, October 2000.

[43] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *Proceedings of the 23th ACM/IEEE Design Automation Conference*, pages 101–107, 1986.

[44] W. C. Wong. Routability optimization with buffer planning in floorplan design. In *Thesis for the Degree of Master of Philosophy in Computer Science and Engineering*, 2002.

[45] J. Xu, P. N. Guo, and C. K. Cheng. Sequence-pair approach for rectilinear module placement. In *IEEE Transactions on CAD of Integrated Circuit and System*, pages 484–493, April 1999.

[46] B. Yao, H. Chen, C. K. Cheng, and R. Graham. Aftermath of floorplan representations. In *Proceedings of Internation Symposium on Physical Design*, pages 138–143, 2001.

[47] E. F. Y. Young, C. C. N. Chu, and Z. C. Chen. Twin binary sequences: A non-reduntant representation for general non-slicing floorplan. In *Proceedings of Internation Symposium on Physical Design*, pages 196–201, 2002.

[48] H. Youssef, S. M. Sait, and K. J. Al-Farra. Timing influenced force directed floorplanning. In *Proceedings of 32th ACM/IEEE Design Automation Conference*, pages 156–161, 1995.

[49] C. X. Zhang, A. Vogt, and D. A. Mlynski. Floorplan design using a hierarchical neural learning algorithm. In *Proceedings of IEEE Internation Symposium on Circuits and Systems*, pages 1664–1667, 1990.

CUHK Libraries
003952793