Routing congestion analysis and reduction in deep sub-micron VLSI design

Zion Cien Shen
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Electrical and Electronics Commons

Recommended Citation

Shen, Zion Cien, "Routing congestion analysis and reduction in deep sub-micron VLSI design " (2004). Retrospective Theses and Dissertations. 1122.
https://lib.dr.iastate.edu/rtd/1122
Routing congestion analysis and reduction in deep sub-micron VLSI design

by

Zion Cien Shen

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Chris Chu, Major Professor
Randall Geiger
Degang Chen
Akhilesh Tyagi
Lu Ruan

Iowa State University
Ames, Iowa
2004
Copyright © Zion Cien Shen, 2004. All rights reserved.
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 3145682
Copyright 2004 by ProQuest Information and Learning Company. All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346
This is to certify that the doctoral dissertation of

Zion Cien Shen

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>1. GENERAL INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Routing Congestion</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Steiner Tree Construction among Blockages</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Dissertation Overview</td>
<td>3</td>
</tr>
<tr>
<td>2. BOUNDS ON THE NUMBERS OF SLICING, MOSAIC AND GENERAL FLOORPLANS</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Contributions</td>
<td>10</td>
</tr>
<tr>
<td>2.3 The Tight Bound for Slicing Floorplan</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 The exact number of skewed slicing trees</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2 The tight bound on the number of slicing floorplans</td>
<td>14</td>
</tr>
<tr>
<td>2.4 The Tight Bound for Mosaic Floorplan</td>
<td>16</td>
</tr>
<tr>
<td>2.5 An Upper Bound for General Floorplan</td>
<td>21</td>
</tr>
<tr>
<td>2.5.1 Empty room insertion</td>
<td>21</td>
</tr>
<tr>
<td>2.5.2 An upper bound on the number of general floorplans</td>
<td>24</td>
</tr>
<tr>
<td>2.6 Conclusion</td>
<td>26</td>
</tr>
<tr>
<td>3. GENERAL FLOORPLAN REPRESENTATION–TWIN BINARY SEQUENCES</td>
<td>27</td>
</tr>
</tbody>
</table>
3.1 Introduction ................................................................. 27
   3.1.1 Previous works ...................................................... 27
   3.1.2 Our contributions .................................................. 29
3.2 Twin Binary Sequences (TBS) Representation .............................. 30
   3.2.1 From floorplan to twin binary trees .............................. 31
   3.2.2 Definition of twin binary sequences ............................. 35
   3.2.3 From TBS to floorplan ......................................... 38
   3.2.4 Size of solution space ........................................ 43
3.3 Extension to General Floorplan ........................................ 44
   3.3.1 Empty room in mosaic floorplan ................................ 44
   3.3.2 Mapping between mosaic floorplan and general non-slicing floorplan 46
   3.3.3 Inserting empty rooms directly on TBS .......................... 49
   3.3.4 Tight bound on the number of irreducible empty rooms ........ 53
3.4 Floorplan Optimization by Simulated Annealing .......................... 54
3.5 Experimental Results .................................................. 59

4. CONGESTION DRIVEN FLOORPLANNING ................................. 63
4.1 Introduction .............................................................. 63
   4.1.1 Previous work ..................................................... 63
   4.1.2 Our contributions ................................................ 65
4.2 Overview of Our Floorplanner .......................................... 67
4.3 Congestion Estimation Model ........................................... 68
   4.3.1 Underlying routing graph ....................................... 68
   4.3.2 Inner dual graph construction from TBS ......................... 70
   4.3.3 Problem formulation ............................................ 72
   4.3.4 Incoming flow balancing (IFB) Phase ............................ 74
   4.3.5 Stepwise flow refinement (SFR) phase .......................... 76
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Global Routing Solution Generation</td>
<td>78</td>
</tr>
<tr>
<td>4.5</td>
<td>Experimental Results</td>
<td>80</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion and Discussion</td>
<td>82</td>
</tr>
<tr>
<td>5.</td>
<td>CONGESTION ESTIMATION MODELS IN PLACEMENT</td>
<td>84</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>84</td>
</tr>
<tr>
<td>5.2</td>
<td>Notations</td>
<td>87</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Lou's model and its revision (<em>LKS</em>)</td>
<td>89</td>
</tr>
<tr>
<td>5.3</td>
<td>Simple Probabilistic Model (<em>SPM</em>)</td>
<td>91</td>
</tr>
<tr>
<td>5.4</td>
<td>Multi-pin Net Probabilistic Model (<em>MPM</em>)</td>
<td>94</td>
</tr>
<tr>
<td>5.5</td>
<td>Post-Probability Processing (<em>PPP</em>)</td>
<td>95</td>
</tr>
<tr>
<td>5.6</td>
<td>Experimental Results</td>
<td>99</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Correlation</td>
<td>100</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Run time</td>
<td>103</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion and Discussion</td>
<td>104</td>
</tr>
<tr>
<td>6.</td>
<td>EFFICIENT RECTILINEAR STEINER TREE CONSTRUCTION</td>
<td>108</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>108</td>
</tr>
<tr>
<td>6.2</td>
<td>Problem Formulation</td>
<td>111</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Rectilinear blockages</td>
<td>111</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Directional blockages</td>
<td>112</td>
</tr>
<tr>
<td>6.3</td>
<td>Escape Graph Versus Spanning Graph</td>
<td>113</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Redundancy in escape graph</td>
<td>113</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Spanning graph</td>
<td>114</td>
</tr>
<tr>
<td>6.4</td>
<td>Spanning Graph Based Approach in <em>RSMTRB</em></td>
<td>115</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Search regions</td>
<td>115</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Spanning graph construction in <em>RSMTRB</em></td>
<td>116</td>
</tr>
</tbody>
</table>
6.5 RMST Construction ........................................... 118
6.6 Experimental Results ........................................ 121
6.7 Conclusion and Discussion ................................... 123
BIBLIOGRAPHY ...................................................... 124
ACKNOWLEDGEMENTS ............................................. 132
LIST OF TABLES

Table 2.1 A summary of results. .......................... 10
Table 3.1 Area minimization. ............................ 61
Table 3.2 Area and wirelength minimization. .......... 61
Table 3.3 Comparisons with ECBL and enhanced Q-sequences. ........ 62
Table 3.4 Comparisons with other representations for slicing floorplan. .... 62
Table 4.1 Experimental results for MCNC benchmarks. ............ 80
Table 4.2 MCNC benchmark. ............................. 81
Table 5.1 Net weight $w(p)$ in MPM. ....................... 96
Table 5.2 Placement Benchmark Statistics .................. 100
Table 5.3 Correlation of predicted congestion with actual results for heavily congested circuits. .................. 105
Table 5.4 Correlation of predicted congestion with actual results for lightly congested circuits. .................. 106
Table 5.5 Run time for congestion estimation models and global router. .... 107
Table 6.1 Statistics of testcases .......................... 121
Table 6.2 Experimental results. ........................... 123
## LIST OF FIGURES

| Figure 2.1 | Relationship among the solution spaces of slicing, mosaic and general floorplans. | 7 |
| Figure 2.2 | Structures that cannot be represented in mosaic floorplan. | 8 |
| Figure 2.3 | Slicing floorplan and its corresponding Skewed Slicing Tree. | 11 |
| Figure 2.4 | A “+”-rooted Skewed Slicing Tree \( T \). | 12 |
| Figure 2.5 | The distribution of \( G(k) \). | 17 |
| Figure 2.6 | An example of reducible and irreducible empty rooms. | 22 |
| Figure 2.7 | A wheel structure. | 22 |
| Figure 2.8 | A pair of \( T \) junctions produce a wheel structure. | 22 |
| Figure 2.9 | An example of inserting maximum number of irreducible empty rooms. | 23 |
| Figure 3.1 | Structures that cannot be represented in a mosaic floorplan. | 29 |
| Figure 3.2 | An example of a twin binary trees. | 31 |
| Figure 3.3 | Building a twin binary trees from a mosaic packing. | 32 |
| Figure 3.4 | Proof of Observation 1 (if part). | 34 |
| Figure 3.5 | Proof of Observation 1 (only if part). | 34 |
| Figure 3.6 | An example of an extended tree. | 35 |
| Figure 3.7 | A simple example of constructing a floorplan from its TBS. | 40 |
| Figure 3.8 | Proof of Theorem 3. | 41 |
| Figure 3.9 | Proof of Theorem 3. | 43 |
Figure 3.10 Examples of reducible and irreducible empty rooms. 44
Figure 3.11 A wheel structure. 45
Figure 3.12 Proof of Lemma 4. 46
Figure 3.13 Proof of Lemma 5. 46
Figure 3.14 Tree structure of an irreducible empty room. 47
Figure 3.15 Mapping between mosaic floorplan and non-slicing floorplan. 48
Figure 3.16 The only two ways to insert $X$ into a tree. 49
Figure 3.17 A simple example of constructing a non-slicing floorplan from a mosaic floorplan. 50
Figure 3.18 An example of searching the last module in the right subtree of $\pi_i$. 52
Figure 3.19 An example of searching the first module in the left subtree of $\pi_i$. 53
Figure 3.20 Floorplan example with many irreducible empty rooms. 55
Figure 3.21 Right-Rotate and Left-Rotate for a binary search tree. 56
Figure 3.22 Modified red-black rotations when subtree $D$ is 0 or 1. 56
Figure 3.23 Four cases of Left-Rotate($T, \pi_i$) on $t_1$. 57
Figure 3.24 Proof of Lemma 7. 59
Figure 4.1 Congestion estimation by the probability approach. 65
Figure 4.2 (a) A rectangular floorplan $R$. (b) Its inner dual graph $G$. The channel segment and the inner dual graph edge corresponding to the adjacent rooms $C$ and $D$ are highlighted. 69
Figure 4.3 An illustration of routing direction assignment. 70
Figure 4.4 An example of routing direction assignment. 70
Figure 4.5 A special case where cycle exists after direction assignment. 71
Figure 4.6 Determining the room neighborhood information directly from TBS. 72
Figure 4.7 Illustration of Incoming Flow Balancing technique. (a) Flow distribution before assigning incoming flow. (b) Flow distribution after assigning incoming flow. 76

Figure 4.8 The convergence of maximum congestion in IFB phase for circuit apte. 76

Figure 4.9 SFR approach to globally optimize the maximum congestion. In this example, e = \{i, j\} is with M_{cong}, V_{out} = \{j, H, L, M, N\}, V_{in} = \{i, C, B, k\}. After one iteration, we derive r_1 = < B, D, E, M > and r_2 = < B, i, j, M >. 77

Figure 4.10 The convergence of M_{cong} of apte in SFR phase, where \( \gamma = 10, \epsilon = 0.001 \). 79

Figure 4.11 Floorplanning result of ami49a using F1. 82

Figure 4.12 Floorplanning result of ami49a using F2. 83

Figure 5.1 Tile structure for congestion estimation 88

Figure 5.2 Proof of theorem 1. 91

Figure 5.3 \( n \) pseudo horizontal routes with weight of 1/n in SPM 92

Figure 5.4 Calculating horizontal usages in SPM 94

Figure 5.5 Congestion estimation by existing probabilistic approach. 96

Figure 5.6 Step one in PPP. (a) The horizontal usages before step one. (b) The horizontal usages after step one. 98

Figure 5.7 Step two in PPP, the over congested horizontal boundary, \( RD_{ij} \), with its six neighbors 98

Figure 5.8 Congestion map by global router. 101

Figure 5.9 Congestion map by LKS. 101

Figure 5.10 Congestion map by MPM. 101

Figure 5.11 Congestion map by SPM. 102
CHAPTER 1. GENERAL INTRODUCTION

The driving force behind the increasingly rapid growth of the VLSI technology has been
the constant shrinking of the feature size of VLSI devices (i.e., the minimum transistor
size). The feature size decreases from about $1\mu m$ in 1990 to $0.09\mu m$ (or 90 nm) early
of 2004. Now, EDA community has been starting to develop tools for 65nm design. It
is also projected that the major chip makers, like Intel and IBM, will be able to put the
65nm chip into production in 2005 using 300nm wafers.

Scaling down of VLSI fabrication into the ultra deep sub-micron (UDSM) dimensions
has dramatic impact on the VLSI technology in several ways. First, the device density on
integrated circuits grows quadratically with the rate of decrease in the feature size. The
total number of transistors on a single chip has increased from 500K in 1985 to 40M today
and will reach 300M in 2006[1]. Second, due to the reduction of wire size and gate size
and increase of wire length, the switching delays in gates decreases, and the signal delays
due to interconnects increase rapidly and have been predominant in today’s designs. All
these make the VLSI design, especially the physical design (which is the back-end of the
VLSI design) more and more challenging.

In this dissertation, the author collects several works done in his Ph.D. years. These
works address several topics related to some practical issues in physical design, like routing
congestion and steiner tree construction among blockages.
1.1 Routing Congestion

The routing congestion can be interpreted as a supply and demand problem for routing resources. The supply of routing resources can be roughly determined by technology parameters, such as die size, number of layers and position of preplaced macros. The demand of routing resources for each design is determined by global routing and detail routing solution.

In traditional design, routing congestion will not be considered until global routing stage. But with growing complexity of chips, routing congestion needs to be emphasized at the planning stages (i.e., floorplanning and placement stages). Since a highly congested region in the floorplan/placement often leads to routing detours around the region which results in a larger routed wire length and worse timing. Congested areas can also deteriorate the performance of global router and, in the worst case, create an unroutable floorplan/placement in the fix-die regime.

Congestion is one of the main optimization objectives in global routing. However, the optimization performance is constrained because the cells are already fixed at this stage. Therefore, designer can save substantial time and resources by detecting and reducing congested regions during the planning stages. An efficient and yet accurate congestion estimation model is crucial to be included in the inner loop of floorplanning and placement design. In this dissertation, we mainly focus on routing congestion modeling and reduction during floorplanning and placement.

1.2 Steiner Tree Construction among Blockages

Given \( n \) points on a plane, a Rectilinear Steiner Minimal Tree (RSMT) connects these points through some extra points called steiner points to achieve a tree with minimal total wire length. Many works have been done on this fundamental problem in VLSI physical design. However, most of them did not take blockages into consideration. In fact, today’s
design often contains many rectilinear routing blockages, e.g., macro cells, IP blocks, and pre-routed nets. Thus, rectilinear Steiner minimal tree construction with rectilinear blockages (RSMTRB), becomes a very practical problem.

Generally, RSMT is used in initial net topology creation for global routing or incremental net tree topology creation in physical synthesis. In addition, RSMT is utilized to accurately estimate congestion and wire length in early design stages, like block floorplanning and cell placement. The timing and congestion information obtained from RSMT can be used as a criteria in timing and congestion driven routing. It is the problem applied hundreds of thousands times and many of them have very large input sizes, RSMT thus deserves much intensive research in VLSI CAD.

Unfortunately, RSMT itself was shown to be strongly NP-complete. Taking blockages into account dramatically increases the problem complexity. Thus, it is extremely unlikely that an efficient optimal algorithm exits for RSMTRB. In the past, several heuristic algorithms were proposed for this problem, however, they have either poor performances or expensive running time. In this dissertation, we propose and efficient and effective graph based approach to solve RSMTRB.

In following section, we give a brief overview of this dissection.

1.3 Dissertation Overview

In Chapter 2, we derive tighter asymptotic bounds on the number of slicing, mosaic and general floorplans. Consider the floorplanning of $n$ blocks. For slicing floorplan, we prove that the exact number is $n^{1\frac{-1^{n+1}}{2}} \sum_{k=0}^{n}(3+\sqrt{8})^{n-2k} \binom{1/2}{k} \binom{1/2}{n-k}$ and the tight bound is $\Theta(n!2^{3n}n^{1.5})$. For mosaic floorplan, we prove that the tight bound is $\Theta(n!2^{3n}n^{4})$. For general floorplan, we prove a tighter lower bound of $\Omega(n!2^{5n}/n^{4})$ and a tighter upper bound of $O(n!2^{5n}/n^{4.5})$. 
In Chapter 3, we proposed a floorplan representation for general non-slicing floorplan. The new representation is called Twin Binary Sequences (TBS), which is the first complete and non-redundant topological representation for non-slicing structure. Like some previous work [34], we have also made used of mosaic floorplan as an intermediate step. However, instead of including a more than sufficient number of extra dummy blocks in the set of modules (that will increase the size of the solution space significantly), our representation allows us to insert an exact number of irreducible empty rooms to a mosaic floorplan such that every non-slicing floorplan can be obtained uniquely from one and only one mosaic floorplan. The size of the solution space is only $O(n!2^{3n/\sqrt{n}1.5})$, since mosaic floorplan is used as an intermediate step, but every non-slicing floorplan can be generated uniquely and efficiently in linear time without any redundant representation.

In Chapter 4, we design an accurate and efficient congestion estimation model by performing global routing. We interpret the global routing problem as a flow problem of several commodities and relax the integral flow constraints. The objective of resulting fractional flow problem is to minimize the maximum congestion over all edges in the inner dual graph. The underlying routing graph for each commodity is derived by assigning directions to the inner dual graph edges. We design an efficient two-phase algorithm to solve this fractional flow problem. The first phase is denoted as Incoming Flow Balancing (IFB) by which a good initial solution is derived. The second phase is called Stepwise Flow Refinement (SFR) by which the maximum congestion of the solution in first phase is iteratively reduced to its optimal value. In addition, a valid global routing solution can be obtained by applying a simple rounding procedure on the fractional flow solution. The maximum congestion after rounding is only increased by 2.82% on average according to our experimental results, which justifies the use of fractional flow to estimate the routing congestion. Finally, we demonstrate our model by integrating it into a simulated annealing (SA) based floorplanner, where we use the maximum congestion as part of the cost of SA. The experimental results show that, on average, our congestion-driven
floorplanner can generate a much less congested floorplan (-36.44%) with a slight sacrifice in area (+1.30%) and wirelength (+2.64%). The runtime of the whole SA process is only increased moderately (+270%).

In Chapter 5, we proposed three congestion estimation models based on probabilistic approach. Particularly, Simple Probabilistic Model (SPM) is proposed by using an efficient probabilistic usage assignment algorithm for two-pin nets. Multi-pin Net Probabilistic Model (MPM) as an extension of SPM, is proposed to directly assign probabilistic usage on multi-pin nets. Post-Probability Processing (PPP) is presented to improve the correlation between the congestion predicted by SPM versus the actual congestion by global router. The experimental results show that MPM is about 21.70 times faster than LKS, a revision of Lou’s model [41], with comparative correlation. SPM with PPP achieves much better correlation than LKS with its run time even less than that of LKS.

In Chapter 6, we propose an efficient and effective approach to construct rectilinear steiner minimum tree with rectilinear blockages. The connection graph we used in this approach is called spanning graph which only contains $O(n)$ edges and vertices. An $O(n \log n)$ time algorithm is proposed to construct spanning graph for RSMTRB. The experimental results shows that this approach can achieve a solution with significantly reduced wire length. The total runtime increased is negligible in the whole design flow.
CHAPTER 2. BOUNDS ON THE NUMBERS OF SLICING, MOSAIC AND GENERAL FLOORPLANS

2.1 Introduction

Floorplanning is a major step in the physical design cycle of VLSI circuits. It is the step to plan the positions and the shapes of the top-level blocks of a hierarchical design. With circuit sizes keep on increasing, floorplanning becomes more and more critical in determining the quality of a layout.

Floorplanning can be viewed as the problem of placing flexible blocks, that is, blocks with fixed area but unknown dimensions. There are many variations in the problem formulation [2, 3, 4]. Unfortunately, all practical floorplanning formulations are NP-complete [2, 3]. As a result, many floorplanners adopt simulated annealing [5] or other stochastic techniques. A code, called a floorplan representation, is usually used to represent the geometrical relationship among the blocks. The code is perturbed repeatedly by the stochastic techniques to search for a good floorplan. The run time and the quality of the solutions depend strongly on the size of the solution space, i.e., the number of possible codes.

The geometrical relationship among the blocks is commonly specified by a rectangular dissection of the floorplan region. The floorplan region is first dissected into rectangular rooms and each block is then mapped to a different room. In order to restrict the size of the solution space, three different ways of dissection are proposed. The corresponding floorplanning structures are called slicing [6], mosaic [7] and general floorplan [8]. Slicing
floorplan is a special case of mosaic floorplan, and mosaic floorplan is a special case of general floorplan. The relationship among the solution spaces of slicing, mosaic and general floorplans is illustrated in Figure 2.1. However, only very loose lower and upper bounds on the size of these three sets are available. The details are discussed below.

![Diagram showing the relationship between Slicing, Mosaic, and General floorplans.]

**Figure 2.1** Relationship among the solution spaces of slicing, mosaic and general floorplans.

Slicing floorplan is a rectangular dissection that can be obtained by recursively cutting a rectangle horizontally or vertically into two smaller rectangles. In [9], Otten first proposed to represent slicing floorplan using a binary tree representation called slicing tree. Each leaf of the slicing tree corresponds to a block and each internal node represents a vertical or horizontal merge operation on the two descendents. Note that one slicing floorplan may correspond to more than one slicing tree. Later, the redundancy was identified by Wong and Liu in [6], where Normalized Polish Expression (NPE) was proposed to represent any slicing structure without redundancy. An upper bound on the number of NPEs, which is also an upper bound on the number of slicing floorplans, is \(O(n!2^{3n}/n^{1.5})\)\(^1\). The best lower bound on the number of slicing floorplans is given by the number of binary trees without labels on internal nodes, which is \(\Omega(n!2^{2n}/n^{1.5})\)[10].

Mosaic floorplan was proposed by Hong et al. in [7]. In mosaic floorplan, non-slicing structures (e.g., a wheel structure) are allowed. However, the floorplan region is dissected into exactly \(n\) rooms so that each room is occupied by one and only one block. In addition,

\(^1\)In this paper, we let \(n\) be the number of blocks in the floorplanning problem.
there is no crossing cut in the mosaic floorplan. See Figure 3.1 for some structures that cannot be represented in mosaic floorplan. Corner Block List (CBL) was proposed in [7] to represent mosaic floorplan. The size of the solution space for CBL is $\Theta(n!2^{3n})$. Notice that some CBLs do not correspond to any floorplan. At about the same time, Sakanushi et al. [11] introduced the Quarter-State Sequence (Q-Sequence) representation for mosaic floorplan. Q-Sequence is a concatenation of room names and two kinds of positional symbols, with the total length equals $3n$. It is a non-redundant representation of mosaic floorplan. An upper bound on the size of the solution space for Q-sequence is $O(n!2^{3n})$. There is no previously available result in literature on the lower bound on the number of mosaic floorplans. So the best lower bound is the same as the one for slicing floorplan.

![Figure 2.2 Structures that cannot be represented in mosaic floorplan.](image)

General floorplan is similar to mosaic floorplan in that non-slicing structures are allowed. However, the floorplan region can be dissected into more than $n$ rooms such that some rooms are not occupied by any block. Many representations have been proposed during the 1990's. In [12], Onodera used Branch-and-Bound algorithm to solve the general floorplan problem. An upper bound on the size of the solution space for this approach is $O(2^{n(n+2)})$, which is extremely huge. In [13], Murata et al. introduced the sequence pair (SP) representation for general floorplan. SP is one of the most elegant representations for general floorplan and has been widely used. Unfortunately, redundancy still exists in this representation. The number of different SP is $\Theta((n!)^2)$. In [14], Nakatake, et al. proposed the Bounded-Sliceline Grid (BSG) representation. In BSG, $n$ blocks are randomly placed...
in a special n-by-n grid. The corresponding size of the solution space is \( O(n!C(n^2, n)) \), which is even larger than that of SP. The huge solution spaces of SP and BSG restrict the applicability of these representations in large floorplan problems. Later, O-tree [15] and B*-tree [16] were proposed to represent a compacted version of general floorplan. Compared to SP and BSG, these two representations have a much smaller solution space of \( \Theta(n!2^{2n}/n^{1.5}) \). However, they represent only partial topological information, and the dimensions of all blocks are required in order to describe an exact floorplan. In addition, not all possible rectangular dissections can be represented by O-tree and B*-tree. For the lower bound on the number of general floorplans, there is no previously available result in literature. So the best lower bound is again the same as the one for slicing floorplan.

Recently, several representations have been proposed to construct general floorplans by inserting empty rooms into mosaic floorplans. They make use of mosaic floorplan as an intermediate step to represent non-slicing structures. In such approach, the number of empty rooms is crucial because it changes the size of the solution space significantly. In [17], Zhou et al. proved that \( n^2 - n \) empty rooms are enough to produce all general floorplans. As a result, the size of the solution space is as huge as \( O(n!C(n^2, n)2^{3n^2}) \). In [18], Zhuang et al. proved that \( n - \lfloor \sqrt{4n - 1} \rfloor \) empty rooms are enough to generate all general floorplans. But the size of the solution space of \( O(2^{6n}(2n)!/n!) \) is still quite large. Recently, Young et al. introduced Twin Binary Sequences (TBS) [19]. TBS is a non-redundant mosaic floorplan representation in which the exact positions for irreducible empty room insertion can be found in linear time. So, by upper-bounding the number of ways to insert empty rooms into each TBS, we can derive an upper bound on the number of general floorplans. We use this idea to derive the bound in Section 2.5.

In [20], Yao et al. showed that the exact number of slicing floorplans is given by the Super Catalan number and the exact number of mosaic floorplans is given by the Baxter number. However, Super Catalan number is given as a recurrence relation and Baxter number is given as a very complicated summation. The growth rate of those numbers
are hard to comprehend. The asymptotic bounds derived in this paper give us a better understanding on those numbers as well as on the number of slicing and mosaic floorplans.

2.2 Contributions

Although many representations of these three types of floorplan have been studied intensively and several upper bounds on the number of combinations of those representations have been reported, it is still theoretically interesting to find the tight bounds on the number of slicing, mosaic and general floorplans. In this paper, we got the exact number of the slicing floorplans is $n!(-1)^{n+1} \sum_{k=0}^{n} (3 + \sqrt{8})^{n-2k} \binom{1/2}{k} \binom{1/2}{n-k}$. Also we prove that the tight bound on this number is $\Theta(n!(3+\sqrt{8})^{n/n^{1.5}}) = \Theta(n!2^{2.543n}/n^{1.5})$. For the number of mosaic floorplans, based on the Baxter number, we show that the tight bound is $\Theta(n!2^{3n}/n^{4})$. For the number of general floorplans, based on the idea of inserting the empty rooms into TBS, we derive a tighter upper bound of $O(n!2^{6n}/n^{4.5})$. Based on the bound for mosaic floorplan, we also get a tighter lower bound of $\Omega(n!2^{3n}/n^{4})$ on the number of general floorplans. The results are summarized in Table 2.1.

<table>
<thead>
<tr>
<th>Previous Bounds</th>
<th>Our Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower Upper</td>
<td>Lower Upper</td>
</tr>
<tr>
<td>Slicing</td>
<td>$\Omega(n!2^{2n}/n^{1.5})$</td>
</tr>
<tr>
<td>Mosaic</td>
<td>$\Omega(n!2^{2n}/n^{1.5})$</td>
</tr>
<tr>
<td>General</td>
<td>$\Omega(n!2^{2n}/n^{1.5})$</td>
</tr>
</tbody>
</table>

These bounds give us a better understanding on the relative sizes of these three types of floorplan. In addition, these bounds could be utilized as a criterion to evaluate the size of the solution space of different floorplan representation.

The organization of this paper is as follows. In Section 2.3, we will show the detailed proof of the exact number and the tight asymptotic bound on the number of slicing
floorplans. In Section 2.4, we will present the tight bound on the number of mosaic floorplans. In Section 2.5, a tighter upper bound on the number of general floorplans will be derived. In Section 2.6, we will conclude the paper.

2.3 The Tight Bound for Slicing Floorplan

In [9], Otten et. al introduced a kind of binary tree called Slicing Tree (ST) to represent slicing structure. A ST is a hierarchical description of the direction of the cuts (vertical or horizontal) in a slicing floorplan. However, for a given slicing floorplan, there may be more than one slicing tree representation. In order to non-redundantly represent all slicing floorplans, Wong and Liu [6] proposed a special kind of slicing tree named Skewed Slicing Tree (SST). A SST is a slicing tree in which no node and its right child have the same label in \{*, +\} (Figure 2.3), where they interpreted the symbols * and + as two binary operators between slicing floorplans. They used the postorder traversal of SST called the Normalized Polish Expression as the floorplan representation. Wong and Liu noted that there is a one-to-one correspondence between the set of NPEs of length \(2n - 1\) and the set of SSTs with \(n\) leaves. Thus, a one-to-one correspondence also exists between all SSTs with \(n\) leaves and all slicing structures with \(n\) rectangular rooms. Therefore, we could obtain the number of slicing floorplan configurations with \(n\) blocks by counting the number of SSTs with \(n\) leaves. Before we explore the tight bound on the number of SSTs, we will first show how the exact number of SSTs can be obtained in the following subsection.

![Figure 2.3 Slicing floorplan and its corresponding Skewed Slicing Tree.](image-url)
2.3.1 The exact number of skewed slicing trees

Suppose there are overall $t_n$ different SSTs with $n$ leaves. When $n = 1$, obviously

$$ t_1 = 1. \quad (2.1) $$

When $n \geq 2$, we classify them into two types of SSTs as follows: $a_n$ "+"-rooted SSTs with $n$ leaves and $b_n$ "*"-rooted SSTs with $n$ leaves. Then

$$ t_n = a_n + b_n. \quad (2.2) $$

Given a "+"-rooted SST $\Gamma$ (see Figure 2.4), according to the definition of SST, the left subtree $L$ of $\Gamma$ could be either a SST ("*"-rooted or "+")-rooted) with fewer than $n$ leaves or a single leaf. The right subtree $R$ of $\Gamma$ could be either a "*"-rooted SST with fewer than $n$ leaves or a single leaf. Then the number of "+"-rooted SSTs with $n$ leaves becomes:

$$ a_n = t_1 b_{n-1} + t_2 b_{n-2} + \cdots + t_{n-2} b_2 + t_{n-1} \cdot 1. \quad (2.3) $$

Similarly, the number of "*"-rooted SSTs with $n$ leaves is

$$ b_n = t_1 a_{n-1} + t_2 a_{n-2} + \cdots + t_{n-2} a_2 + t_{n-1} \cdot 1. \quad (2.4) $$

![Figure 2.4 A "+"-rooted Skewed Slicing Tree $\Gamma$.](image)

According to Eq. (2.2), (2.3) and (2.4), the number of SSTs with $n$ leaves becomes

$$ t_n = t_1 (a_{n-1} + b_{n-1}) + t_2 (a_{n-2} + b_{n-2}) + \cdots + t_{n-2} (a_2 + b_2) + 2t_{n-1} + t_{n-1} = t_1 t_{n-1} + t_2 t_{n-2} + \cdots + t_{n-2} t_2 + t_{n-1} t_1 + t_{n-1}. $$
In order to solve the recurrence equation (2.5) with the initial condition (5.2.1), we define the generating function as

\[ T(z) = t_1 + t_2z + t_3z^2 + \cdots \] (2.6)

Then, we have

\[ T^2(z) = t_1^2 + (t_1t_2 + t_2t_1)z + (t_1t_3 + t_2t_2 + t_3t_1)z^2 + \cdots \]

and so

\[ T^2(z) + T(z) = (t_1^2 + t_1) + (t_1t_2 + t_2t_1 + t_2)z + (t_1t_3 + t_2t_2 + t_3t_1 + t_3)z^2 + \cdots \] (2.7)

Combining (2.6) and (2.7) yields

\[ t_1 + [T^2(z) + T(z)]z = T(z). \]

Since \( t_1 = 1 \), then

\[ zT(z)^2 + (z - 1)T(z) + 1 = 0. \] (2.8)

Solving Eq. (2.8) with initial condition \( T(0) = t_1 = 1 \) yields

\[ T(z) = \frac{1 - z - \sqrt{z^2 - 6z + 1}}{2z}. \]
Let $\alpha = 3 + \sqrt{8}$ and $\beta = 3 - \sqrt{8}$. Notice that $\alpha \beta = 1$. Thus

$$T(z) = \frac{1 - z - \sqrt{\alpha - z\sqrt{\beta - z}}}{2z}$$

$$= \frac{1}{2z} \left[ 1 - z - \sqrt{\frac{1 - z}{\alpha} \sqrt{1 - z}} \right]$$

$$= \frac{1}{2z} \left[ 1 - z \sum_{i=0}^{\infty} \left( \frac{1}{i} \right) \left( -\frac{1}{\alpha} \right)^i z^i \cdot \sum_{j=0}^{\infty} \left( \frac{1}{j} \right) \left( -\frac{1}{\beta} \right)^j z^j \right]$$

By the definition of generating function (2.6), for $n \geq 2$, we get the coefficient of $z^{n-1}$ as

$$t_n = -\frac{1}{2} \sum_{k=0}^{n} \binom{\frac{1}{2}}{k} \left( \frac{-1}{\alpha} \right)^k \left( \frac{1}{2} \right)^{n-k} \left( \frac{1}{\beta} \right)^{n-k}$$

$$= \frac{(-1)^{n+1}}{2} \sum_{k=0}^{n} \alpha^{n-2k} \binom{\frac{1}{2}}{k} \left( \frac{1}{2} \right)^{n-k}$$

(2.9)

The exact number of SSTs thus has the general form

$$t_n = \begin{cases} 
1 & \text{if } n = 1 \\
\frac{(-1)^{n+1}}{2} \sum_{k=0}^{n} \alpha^{n-2k} \binom{\frac{1}{2}}{k} \left( \frac{1}{2} \right)^{n-k} & \text{if } n \geq 2
\end{cases}$$

In the following subsection, we try to get the tight bound on $t_n$.

2.3.2 The tight bound on the number of slicing floorplans

In order to obtain the tight bound on $t_n$ ($n \geq 2$), we rewrite Eq. (2.9) as

$$t_n = \sum_{k=0}^{n} F(k)$$

where

$$F(k) = \frac{(-1)^{n+1}}{2} \alpha^{n-2k} \binom{\frac{1}{2}}{k} \left( \frac{1}{2} \right)^{n-k}$$
First, we bound $F(0)$ as follows:

$$F(0) = \frac{(-1)^{n+1}}{2} \alpha^n \left( \frac{1}{2} \right)_n$$

$$= \frac{(-1)^{n+1}}{2} \alpha^n \frac{1}{2} \frac{1}{2} - (1 - n + 1)$$

$$= \frac{\alpha^n (1 \times 3 \times \cdots \times (2n-3))}{2^{n+1} n!}$$

$$= \frac{\alpha^n (2n)!}{2^{2n+1} (n!)^2 (2n-1)}$$

then, by Stirling’s approximation\(^2\)

$$= \Theta\left( \frac{\alpha^n \sqrt{2\pi n} (\frac{2n}{e})^{2n}}{2^{2n+1} (\sqrt{\pi n})^2 (\frac{2n}{e})^{2n} (2n-1)} \right)$$

$$= \Theta(\alpha^n / n^{1.5})$$

$$= \Theta(2.54n / n^{1.5})$$

Second, we will bound $\sum_{k=1}^{n-1} F(k)$. For $1 \leq k \leq n - 1$, let

$$r_k = \frac{F(k)}{F(k-1)} = \frac{(n - k + 1)(k - \frac{3}{2})}{\alpha^2 (n - k - \frac{1}{2})k}.$$

When $n \to \infty$, it is not difficult to observe that

$$r_1 \approx -0.0147,$$

$$0 < r_2 < r_3 < \cdots < r_{n-1} < 1,$$

$$r_{n-1} \approx 0.1177.$$

Therefore, $F(1) = r_1 F(0) < 0$ and

$$\sum_{k=1}^{n-1} F(k) = F(1) + F(1) r_2 + F(1) r_2 r_3 + \cdots + F(1) r_2 \cdots r_{n-1}$$

$$> F(1) + F(1) r_{n-1} + F(1) r_{n-1}^2 + \cdots + F(1) r_{n-1}^{n-2}$$

$$> \frac{F(1)}{1 - r_{n-1}}$$

$$\approx -0.0166 F(0).$$

\(^2\text{Stirling’s approximation for } n! = \Theta\left( \sqrt{2\pi n} \left( \frac{n}{e} \right)^n \right) [10].\)
Thus we bound \( \sum_{k=1}^{n-1} F(k) \) as

\[-0.0166 F(0) < \sum_{k=1}^{n-1} F(k) < 0.\]

Third, we bound \( F(n) \) as

\[F(n) = \alpha^{-2n} F(0) = o(F(0)).\]

We thus get the tight bound on \( t_n \) as

\[t_n = F(0) - 0.0166 F(0) + o(F(0)) = \Theta(F(0)) = \Theta(2^{2.543n}/n^{1.5})\]

If we consider the labels of the leaves of SSTs, there are \( n! \) combinations for labeling of the leaves of SSTs. Thus the total number of combinations of slicing floorplan is \( \Theta(n!2^{2.543n}/n^{1.5}) \).

### 2.4 The Tight Bound for Mosaic Floorplan

In paper [20], Yao et al. first proved that the number of combinations of mosaic floorplan with \( n \) blocks is equal to the number of Baxter permutations on \( \{1, \ldots, n\} \). Then, \( M(n) = B(n) \), where \( M(n) \) is the exact number of combinations of mosaic floorplans with \( n \) blocks, and \( B(n) \) is a Baxter number, which can be represented as follows:

\[
B(n) = \left( \begin{array}{c} n+1 \\ 1 \end{array} \right)^{-1} \left( \begin{array}{c} n+1 \\ 2 \end{array} \right)^{-1} \sum_{k=1}^{n} \left( \begin{array}{c} n+1 \\ k-1 \end{array} \right) \left( \begin{array}{c} n+1 \\ k \end{array} \right) \left( \begin{array}{c} n+1 \\ k+1 \end{array} \right)
\]  

(2.10)
In order to get the tight bound on $B(n)$, we first simplify Eq. (2.10) as follows:

$$B(n) = \frac{2}{n(n+1)^2} \sum_{k=1}^{n} \frac{((n+1)!)^3}{(k-1)!(n-k+2)!k!(n-k+1)!(k+1)!(n-k)!}$$

$$= \sum_{k=1}^{n} \frac{2(n!)^3(n+1)}{n(k-1)!k!(k+1)!(n-k)!(n-k+1)!(n-k+2)!}$$

$$= \sum_{k=1}^{n} G(k),$$

where

$$G(k) = \frac{2(n!)^3(n+1)}{n(k-1)!k!(k+1)!(n-k)!(n-k+1)!(n-k+2)!}$$

Without lost of generality, we assume $n$ is even. Note that for $1 \leq k \leq n$

$$G(k) = G(n-k+1)$$

and for $2 \leq k \leq \frac{n}{2}$

$$\frac{G(k)}{G(k-1)} = \frac{(n-k+1)(n-k+2)(n-k+3)}{(k-1)k(k+1)} > 1$$

Therefore, for $1 \leq k \leq \frac{n}{2}$, $G(k)$ will increase with the increasing of $k$; for $\frac{n}{2} + 1 \leq k \leq n$, $G(k)$ will decrease with the increasing of $k$. Thus, the distribution of $G(k)$ will be roughly like Figure 2.5.

Figure 2.5  The distribution of $G(k)$. 
First, we bound $G\left(\frac{n}{2}\right)$ as

$$
G\left(\frac{n}{2}\right) = \frac{2(n!)^3(n+1)}{n(n-1)!\left(\frac{n}{2}\right)!\left(\frac{n}{2}+1\right)!\left(\frac{n}{2}+2\right)! (n!)^3(n+1)}
= \Theta\left(\frac{(2\pi n)^{3/2}(\frac{n}{e})^{3n}}{(\pi n)^3(\frac{n}{2e})^{3n}n^3}\right)
= \Theta\left(\frac{2^{3n}}{n^{4.5}}\right).
$$

Second, we try to bound $\sum_{k=1}^{\frac{n}{2}} G(k)$.

Lemma 1. For $\frac{n}{2} - \lfloor \sqrt{n \ln n} \rfloor \leq k \leq \frac{n}{2}$, when $n \to \infty$, $G(k) = \frac{G\left(\frac{n}{2}\right)}{M}$, where $M = \Theta\left((\frac{2}{n})^{3k}(2 - \frac{2k}{n})^{3(n-k)}\right)$.

Proof: For $\frac{n}{2} - \lfloor \sqrt{n \ln n} \rfloor \leq k \leq \frac{n}{2}$, when $n \to \infty$, notice that $k \to \infty$, $n-k \to \infty$. By Stirling’s approximation, we work out $G(k)$ as

$$
G(k) = \Theta\left(\frac{(n!)^3}{n^3(k!)^3((n-k)!)^3}\right)
= \Theta\left(\frac{(2\pi n)^{3/2}(\frac{n}{e})^{3n}}{(2\pi k)^{3/2}(\frac{k}{e})^{3k}(2\pi(n-k))^{3/2}(\frac{n-k}{e})^{3(n-k)}n^{3n}}\right)
= \Theta\left(\frac{n^{4.5}k^{3k}(n-k)^{3(n-k)}}{2^{3n}}\right)
= \Theta\left(\frac{G\left(\frac{n}{2}\right)}{M}\right),
$$

where $M = \Theta\left((\frac{2}{n})^{3k}(2 - \frac{2k}{n})^{3(n-k)}\right)$.

Lemma 2. When $k = \frac{n}{2} - \lfloor \frac{1}{\sqrt{12}} \sqrt{n \ln n} \rfloor$, $G(k) = \Theta\left(n^{-1}\right) G\left(\frac{n}{2}\right)$.

Proof: Assume

$$
k = \frac{n}{2} - \lfloor c \sqrt{n \ln n} \rfloor
$$

(2.12)
where $0 \leq c \leq 1$. By Lemma 1, we have

$$M = \Theta \left( \left( 1 - 2c \sqrt{\frac{\ln n}{n}} \right)^{3\left(\frac{1}{2} - c\sqrt{n\ln n}\right)} \left( 1 + 2c \sqrt{\frac{\ln n}{n}} \right)^{3\left(\frac{1}{2} + c\sqrt{n\ln n}\right)} \right).$$

Using the limit of function

$$\lim_{x \to 0} (1 \pm x)^k = e^{\pm k},$$

we simplify $M$ as

$$M = \Theta \left( e^{-\left(3c\sqrt{n\ln n} - 6c^2\ln n\right)} \cdot e^{(3c\sqrt{n\ln n} + 6c^2\ln n)} \right) = \Theta \left( n^{12c^2} \right).$$

Let $c = \frac{1}{\sqrt{12}}$, by (2.11) and (2.13), we get

$$G \left( \frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor \right) = \Theta \left( n^{-1} \right) G \left( \frac{n}{2} \right)$$

Noticing that

$$G(1) < G(2) < \cdots < G \left( \frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor \right),$$

we thus bound $\sum_{k=1}^{\frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor} G(k)$ as follows:

$$0 < \sum_{k=1}^{\frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor} G(k) < \left( \frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor \right) \Theta(n^{-1}) G \left( \frac{n}{2} \right)$$

$$= O \left( G \left( \frac{n}{2} \right) \right)$$

Lemma 3. For $\frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor \leq k \leq \frac{n}{2}$,

$$\sum_{k=\frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor}^{\frac{n}{2}} G(k) \text{ is bounded by } \Theta \left( \sqrt{n} G \left( \frac{n}{2} \right) \right).$$

Proof: For $\frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n\ln n} \right\rfloor \leq k \leq \frac{n}{2}$, we take $G(k)$ as an continuous function. Thus we bound the summation of $G(k)$ by bounding the integration of continuous function $G(k)$
where \( \frac{n}{2} - \frac{1}{\sqrt{12}} \sqrt{n \ln n} \leq k \leq \frac{n}{2} \). By (2.11), (2.12) and (2.13)

\[
\sum_{k=\frac{n}{2} - \left[ \frac{1}{\sqrt{12}} \sqrt{n \ln n} \right]}^{\frac{n}{2}} G(k) = \Theta \left( \int_{\frac{n}{2} - \left[ \frac{1}{\sqrt{12}} \sqrt{n \ln n} \right]}^{\frac{n}{2}} G(k) \, dk \right)
\]

\[
= \sqrt{n \ln n} G\left( \frac{n}{2} \right) \Theta \left( \int_{\frac{1}{\sqrt{12}}}^{0} -n^{-12c^2} \, dc \right)
\]

\[
= \sqrt{n \ln n} G\left( \frac{n}{2} \right) \Theta \left( \int_{0}^{\frac{1}{\sqrt{12}}} n^{-12c^2} \, dc \right).
\]

In order to solve the integration \( \int_{0}^{\frac{1}{\sqrt{12}}} n^{-12c^2} \, dc \), we use the property of Normal Distribution with the probability density function as

\[
P(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}
\]

for \( 0 \leq x \leq 3\sigma \)

\[
\frac{1}{\sigma \sqrt{2\pi}} \int_{0}^{3\sigma} e^{-\frac{x^2}{2\sigma^2}} \, dx \approx 0.5
\]

i.e.

\[
\int_{0}^{3\sigma} e^{-\frac{x^2}{2\sigma^2}} \, dx \approx \sqrt{\frac{\pi}{2\sigma}}
\]

Then

\[
\int_{0}^{\frac{1}{\sqrt{12}}} n^{-12c^2} \, dc = \int_{0}^{\frac{1}{\sqrt{12}}} e^{-12c^2 \ln n} \, dc
\]

Notice that when \( n \to \infty \)

\[
\frac{1}{\sqrt{12}} \gg 3 \cdot \frac{1}{\sqrt{24 \ln n}},
\]

thus

\[
\int_{0}^{\frac{1}{\sqrt{12}}} n^{-12c^2} \, dc \approx \sqrt{\frac{\pi}{2}} \frac{1}{\sqrt{24 \ln n}}.
\]

Then, we have

\[
\sum_{k=\frac{n}{2} - \left[ \frac{1}{\sqrt{12}} \sqrt{n \ln n} \right]}^{\frac{n}{2}} G(k) = \Theta(G\left( \sqrt{n} G\left( \frac{n}{2} \right) \right))
\]
We thus obtain the tight bound on $B(n)$ as

$$B(n) = \sum_{k=1}^{n} G(k)$$

$$= 2 \left[ \sum_{k=1}^{\frac{n}{2} - \left\lfloor \sqrt{\frac{n \ln n}{12}} \right\rfloor} G(k) + \sum_{k=\left\lfloor \sqrt{\frac{n \ln n}{12}} \right\rfloor}^{\frac{n}{2}} G(k) \right]$$

$$- 2G\left(\frac{n}{2} - \left\lfloor \frac{1}{\sqrt{12}} \sqrt{n \ln n} \right\rfloor\right)$$

$$= 2\left[ \Theta\left(G\left(\frac{n}{2}\right)\right) + \Theta\left(\sqrt{n}G\left(\frac{n}{2}\right)\right) - \Theta(n^{-1})G\left(\frac{n}{2}\right) \right]$$

$$= \Theta\left(\sqrt{n}G\left(\frac{n}{2}\right)\right)$$

$$= \Theta\left(2^{3n}/n^4\right).$$

If we consider the labeling of block names, the final tight bound on the number of mosaic floorplans with $n$ blocks is $\Theta(n!2^{3n}/n^4)$.

### 2.5 An Upper Bound for General Floorplan

In paper [19], a general floorplan $F'$ can be constructed from a mosaic floorplan $F$ by inserting some irreducible empty rooms into a mosaic floorplan at right places in $F$. There are two kinds of empty rooms. One is called reducible empty room which is resulted from assigning a small block into a big room (see an example in Figure 2.6(a)). Another kind of empty room is called irreducible empty rooms which can not be removed by merging with another room in the packing (see an example in Figure 2.6(b)). In addition, a wheel structure always exists in every irreducible empty room (see Figure 3.11). We discuss this idea in more detail in the following subsection.

#### 2.5.1 Empty room insertion

Observation 1. A wheel structure can be produced from and only from the following mosaic structure: a pair of T junctions share the same channel on each side, respectively.
Figure 2.6 An example of reducible and irreducible empty rooms.

Figure 2.7 A wheel structure.

It is shown in Figure 2.8.

Based on the Observation 1, it is not difficult to prove the following Lemma.

Lemma 4. For a channel with $p$ and $q$ blocks on each side, respectively, the maximum number of irreducible empty rooms which could be inserted along the channel is $\min(p, q) - 1$.

Proof: Without lost of generality, we assume $p \leq q$. Then, from $p$ blocks on one side of a channel, we could find out $p - 1$ T junctions. Similarly, for $q$ blocks along the other side of the channel, there is $q - 1$ T junctions. Therefore, at most we could pick $p$ pairs of T junctions from each side at one time. According to the Observation 1, an empty room could be produced by any pair of T junctions with one from each side of the
channel, respectively. We label these T junctions as $T_1, T_2, \cdots, T_{p-1}$ on each side from top to bottom or from left to right, and then match them one by one according to the order from $T_1$ to $T_p$. $p-1$ empty rooms could thus be inserted along the channel. An example with 4 blocks and 5 blocks along a channel $c$ on each side, respectively, is shown in Figure 2.9. Maximumly, 3 irreducible empty rooms (represented by X) are inserted in this example.

![Figure 2.9](image)

**Figure 2.9** An example of inserting maximum number of irreducible empty rooms.

**Lemma 5.** For a channel with $p$ and $q$ blocks on each side, respectively, the number of ways to insert empty rooms along the channel is

$$\binom{p + q - 2}{q - 1}.$$  

**Proof:** Without lost of generality, we assume $p \leq q$. According to Lemma 4, we can insert at most $p-1$ empty rooms along the channel, which means $0 \leq j \leq p-1$ empty rooms could possibly inserted along the channel. Similar to the PROOF of Lemma 4, we pick $j$ pairs of $T$ junctions from each side of the channel, and label them from top to bottom or left to right and then match them one by one according to the order. We can thus insert $j$ empty rooms into those $T$ junctions.

Let $C(p, q)$ denote the number of ways to insert empty rooms along the channel with
p and q blocks on each side, respectively. We have

\[
C(p, q) = \sum_{j=0}^{p-1} \binom{p-1}{j} \binom{q-1}{j}
\]

\[
= \sum_{j=0}^{p-1} \binom{p-1}{p-1-j} \binom{q-1}{j}
\]

\[
= \binom{p+q-2}{p-1} \quad \text{by the definition of Combination}
\]

\[
= \binom{p+q-2}{q-1}
\]

In the next subsection, we will formulate the total number of ways to insert empty rooms into a mosaic floorplan with n blocks.

2.5.2 An upper bound on the number of general floorplans

Given a mosaic floorplan with n blocks, by counting the total number of ways to insert empty rooms into the mosaic floorplan, we can obtain the total number of general floorplan generated from the mosaic floorplan.

For a mosaic floorplan with n blocks, it has overall n + 3 channels. Without lost of generality, we assume it has k (2 ≤ k ≤ n + 1) horizontal channels with the uppermost boundary as the 1st horizontal channel and downmost boundary as kth horizontal channel. In addition, it has n + 3 − k vertical channels with the leftmost boundary as 1st vertical channel and rightmost boundary as (n + 3 − k)th vertical channel.

Let \( h_i \) (i = 1, 2, ⋯ , k) be the number of blocks which touch the \( i^{th} \) horizontal channel on the top, \( h'_i \) be the number of blocks which touch the \( i^{th} \) horizontal channel on the bottom. Let \( v_j \) (1 ≤ j ≤ n + 3 − k) be the number of blocks which touch the \( j^{th} \) vertical channel on
the left, \( v'_j \) be the number of blocks which touch the \( j^{th} \) vertical channel on the right. Assuming \( h_1 = h'_k = v_1 = v'_{n+3-k} = 1 \), we have

\[
\sum_{i=1}^{k} h_i = \sum_{i=1}^{k} h'_i = \sum_{j=1}^{n+3-k} v_j = \sum_{j=1}^{n+3-k} v'_j = n + 1.
\]

We denote \( L(n) \) as the total number of ways to insert empty rooms into a mosaic floorplan with \( n \) blocks, according to Lemma 5

\[
L(n) = \prod_{i=1}^{k} C(h_i, h'_i) \cdot \prod_{j=1}^{n+3-k} C(v_j, v'_j)
\]

(2.14)

In order to get an upper bound on \( L(n) \), we notice that \( \forall p, q, s, t \in \{0, 1, 2, \cdots, n\} \)

\[
\begin{pmatrix} p \\ q \end{pmatrix} \cdot \begin{pmatrix} s \\ t \end{pmatrix} \leq \begin{pmatrix} p + s \\ q + t \end{pmatrix}
\]

We thus bound (2.14) as

\[
L(n) \leq \left( \sum_{i=1}^{k} (h_i + h'_i - 2) \right) \cdot \left( \sum_{j=1}^{n+3-k} (v_j + v'_j - 2) \right)
\]

\[
= \left( \frac{2n + 2 - 2k}{n + 1 - k} \right) \cdot \left( \frac{2n + 2 - 2(n + 3 - k)}{n + 1 - (n + 3 - k)} \right)
\]

\[
\leq \left( \frac{2n - 2}{n - 1} \right) \cdot \frac{(2n - 2)!}{(n - 1)!^2}
\]

then, by Stirling’s approximation

\[
= O \left( \frac{\sqrt{2\pi(2n - 2)}((2n - 2)/e)^{2n-2}}{(\sqrt{2\pi(n - 1)})^2((n - 1)/e)^{2n-2}} \right)
\]

\[
= O(2^{2n}/n^{0.5})
\]

Since the tight bound on the number of mosaic floorplan with \( n \) blocks is \( \Theta(n!2^{3n}/n^4) \), we obtain an upper bound on the number of general floorplan with \( n \) blocks as \( O(n!2^{5n}/n^{4.5}) \).
2.6 Conclusion

We have successfully obtained tight bounds of $\Theta(n!2^{2.543n}/n^{1.5})$ on the number of slicing floorplans and $\Theta(n!2^{3n}/n^{4})$ on the number of mosaic floorplans. However, for the number of general floorplans, the lower bound $\Omega(n!2^{3n}/n^{4})$ is still significantly smaller than the upper bound $O(n!2^{5n}/n^{4.5})$. We will work on the tight bound on the number of general floorplans in the future.

Regarding floorplan representations, NPE is a non-redundant representation for slicing floorplan. Q-sequence and TBS are two non-redundant representations for mosaic floorplan. However, there is no non-redundant representation for general floorplan. Although all general floorplans can be produced by inserting empty rooms into TBSs, the information describing which empty room to be inserted is not uniform. Hence TBS cannot be easily extended to a succinct representation which describes a general floorplan completely. We will also work on the problem of designing an elegant and non-redundant general floorplan representation in the future.
CHAPTER 3. GENERAL FLOORPLAN REPRESENTATION−TWIN BINARY SEQUENCES

3.1 Introduction

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

3.1.1 Previous works

The problem of floorplan representation has been studied extensively. There are three types of floorplan: slicing, mosaic and non-slicing. A slicing floorplan is a floorplan that can be obtained by recursively cutting a rectangle into two by using a vertical line
or a horizontal line. Normalized Polish expression [32] is the most popular method to represent slicing floorplan. This representation can describe any slicing structure with no redundancy. An upper bound on its solution space is $O(n!2^{3n-3}/n^{1.5})$. For general floorplan that is not necessarily slicing, there was no efficient representation other than the constraint graphs until the sequence pair (SP) [27] and the bounded-sliceline grid (BSG) [28] appeared in the mid 90’s. The SP representation has been widely used because of its simplicity. Unfortunately, there are a lot of redundancies in these representations. The size of the solution space of SP is $(n!)^2$ and that of BSG is $n!C(n^2, n)$. This drawback has restricted the applicability of these methods in large scale problems. O-tree [24] and B*-tree [21] are later proposed to represent compacted (admissible) non-slicing floorplan. They have very small solution space of $O(n!2^{2n-2}/n^{1.5})$ and can give a floorplan in linear time. However, they describe only partial topological information and module dimensions are needed to give a floorplan exactly. The representation is not unique, and a single O-tree or B*-tree representation, depending on the module dimensions, can lead to more than one floorplan with modules of different topological relationships with each other.

The paper [25] proposes a new type of floorplan called mosaic floorplan. A mosaic floorplan is similar to a general non-slicing floorplan except that it does not have any unoccupied room (Figure 3.1(a)) and there is no crossing cut in the floorplan (Figure 3.1(b)). A representation called Corner Block List (CBL) is proposed to represent mosaic floorplan. This representation has a relatively small solution space of $O(n!2^{3n})$ \(^1\) and the time complexity to realize a floorplan from its representation is linear. However, some corner block lists do not correspond to any floorplan. As a remedy to the weakness that some non-slicing structures cannot be represented (e.g., Figure 3.1(a)), CBL is extended by including dummy blocks of zero area in the set of modules. In order to represent all

\(^1\)In [25], the paper claims without proof that the size of solution space for Corner Block List is $O(n!2^{3n}/n^{1.5})$. However, we believe that the correct size of CBL solution space should be $\Theta(n!2^{3n})$. In the CBL algorithm, the corner block list $(S, L, T)$ are perturbed randomly and independently in the simulated annealing process. There are $n!$ combinations for $S$, $2^{n-1}$ combinations for $L$, and $2^{2n-3}$ combinations for $T$. So the total number of combinations is $\Theta(n!2^{3n})$.\)
non-slicing structure, $O(n^2)$ of such dummy blocks are used and this has increased the size of the solution space significantly [34]. In the paper [30], a new representation called Q-sequence is proposed to represent mosaic floorplan, which is later enhanced in the paper [35] by including empty rooms. It is also proved in [35] that the number of empty rooms required is upper bounded by $n - \lfloor \sqrt{4n - 1} \rfloor$ where $n$ is the number of modules.

\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline
C & D \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
A & B \\
\hline
C & D \\
\hline
\end{array}
\]

(a) (b)

Figure 3.1 Structures that cannot be represented in a mosaic floorplan.

3.1.2 Our contributions

Although the problem of floorplan representation has been studied extensively, and numerous floorplan representations have been proposed in recent years, it is still practically useful and theoretically interesting to find a complete (i.e., every non-slicing floorplan can be represented) and non-redundant topological representation for general non-slicing structure. In this chapter, we will present such a representation, the Twin Binary Sequences. This will mark the first of this kind. Like some previous work [34], we have made use of mosaic floorplan as an intermediate step to represent a non-slicing structure. However, instead of including an extra number of dummy blocks in the set of modules, the representation allows us to insert an exact number of irreducible empty rooms to a mosaic floorplan such that every non-slicing structure can be generated uniquely and non-redundantly. Besides, the representation can give a floorplan efficiently in linear time. We have studied the relationship between mosaic and non-slicing floorplan and have proved
that the number of empty rooms needed to be inserted into a mosaic floorplan to obtain a non-slicing structure is tightly bounded by $\Theta(n)$ where $n$ is the number of modules. \footnote{Together with the upper bound result in [35], the tight bound can be further improved to $\Theta(n - 2\sqrt{n})$.}

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

### 3.2 Twin Binary Sequences (TBS) Representation

In the paper [33], Yao, et al. first suggest that the Twin Binary Trees (TBT) can be used to represent mosaic floorplan. They have shown a one-to-one mapping between mosaic floorplan and TBT. We have made use of TBT in our representation. Recall that the definition of twin binary trees comes originally from the paper [23] as follows:

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

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

where $\text{Tree}_n$ is the set of binary trees with $n$ nodes, and $\Theta(b)$ is the labeling of a binary tree $b$ as follows. Starting with an empty sequence, we perform an inorder traversal of the tree $b$. When a node with no left child is reached, we will add a bit 0 to the sequence, and when a node with no right child is reached, we will add a bit 1 to the sequence. The first 0 and the last 1 will be omitted. $\Theta^c$ is the complement of $\Theta$ obtained by interchanging all the 0's and 1's in $\Theta$. An example of a twin binary trees is shown in Figure 3.2.

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

3.2.1 From floorplan to twin binary trees

Given a mosaic floorplan \( F \), we can obtain a pair of twin binary trees \( t_1 \) and \( t_2 \) by traveling along the slicelines of \( F \). An example is shown in Figure 4.6. To construct \( t_1 \), we start from the module at the lower left corner and travel upward (left subtree) and to the right (right subtree). Whenever the lower left corner of another module \( x \) is reached, a node labeled \( x \) will be inserted into the tree and the process will be repeated starting from module \( x \) until all the modules in the floorplan are visited. The tree \( t_2 \)
can be built similarly by starting from the module at the upper right corner and travel downward (right subtree) and to the left (left subtree). Similarly, whenever the upper right corner of another module \( y \) is reached, a node labeled \( y \) will be inserted into the tree and the process will be repeated starting from \( y \) until all the modules are visited.

The paper [33] has shown that the pair of trees built in this way must be twin binary to each other, and there is a one-to-one mapping between mosaic floorplan and twin binary trees. We observed that the inorder traversal of the two binary trees constructed by the above method must be the same. Let us look at the example in Figure 4.6. We can see that the inorder traversals of both \( t_1 \) and \( t_2 \) are \( ABCFDE \). We have proved the following observation that helps in defining the Twin Binary Sequences representation:

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

**Proof:** *(if part)* This part can be proved by induction on the number of modules in the floorplan. The base case occurs when there is only one module in the floorplan and conditions (1) and (2) follow trivially. Assume that these conditions are true when there are not more than \( k \geq 1 \) modules in the floorplan. Consider a floorplan \( F \) with \( k + 1 \)
modules. Let the pair of binary trees constructed from $F$ by the above method be $t_1$ and $t_2$. Consider the module $m$ at the upper left corner of $F$. There are only four possible configurations for the position of $m$ in $F$ as shown in Figure 3.4. In each case, let $F'$ be the floorplan obtained by sliding module $m$ out of $F$ by moving the thickened sliceline in the direction shown. Let $t_1'$ and $t_2'$ be the pair of binary trees constructed from $F'$ by the above method. Since floorplan $F'$ has only $k$ modules, $t_1'$ and $t_2'$ satisfy conditions (1) and (2) according to the hypothesis, i.e., $\Theta(t_1') = \Theta(t_2')$, and their inorder traversals are the same. From Figure 3.4, we can see that in case (a) and (c), $\Theta(t_1) = 1\Theta(t_1'), \Theta(t_2) = 0\Theta(t_2')$ and the inorder traversal of $t_1$ ($t_2$) is the same as that obtained by appending $m$ in front of the inorder traversal of $t_1'$ ($t_2'$). Similarly, in case (b) and (d), $\Theta(t_1) = 0\Theta(t_1'), \Theta(t_2) = 1\Theta(t_2')$ and the inorder traversal of $t_1$ ($t_2$) is the same as that obtained by appending $m$ in front of the inorder traversal of $t_1'$ ($t_2'$). Therefore $t_1$ and $t_2$ also satisfy conditions (1) and (2).

(only if part) Again, this part is proved by induction. The base case occurs when there is only one node in the pair of binary trees. If both conditions (1) and (2) are true (note that condition (1) must be true since there is only one node in the trees and their labelings are both empty), their nodes are labeled the same and they correspond to a packing with only one module. Assume that this statement is true for any pair of trees with $k \geq 1$ nodes, i.e., inorder traversal of length $k$ and labeling of length $k - 1$. Consider a pair of trees ($t_1$ and $t_2$) with inorder traversal $m_1m_2 \ldots m_{k+1}$, and labelings $b_1b_2 \ldots b_k$ and $\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_k$. There are two cases as shown in Figure 3.5 according to the value of the bit $b_1$. In both cases, the inorder traversal $m_2m_3 \ldots m_{k+1}$, and the bit sequences $b_2b_3 \ldots b_k$ and $\bar{b}_2, \bar{b}_3, \ldots, \bar{b}_k$ will correspond to a floorplan $F'$ according to the hypothesis. We can obtain a floorplan $F$ from $F'$ by putting the module $m_1$ on the right (case (a)) or at the top (case (b)). $F$ will correspond to a pair of trees with inorder traversal $m_1m_2 \ldots m_{k+1}$, and labelings $b_1b_2 \ldots b_k$ and $\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_k$. We can choose between case (a) and (b) depending on the value of $b_1$. Therefore this only if statement is also true when there are $k+1$ nodes in the pair of trees.
Figure 3.4 Proof of Observation 1 (if part).

Figure 3.5 Proof of Observation 1 (only if part).
If we extend a tree $t$ by adding a left child of bit 0 to every node (except the leftmost node) that has no left child and by adding a right child of bit 1 to every node (except the rightmost node) that has no right child, the tree obtained is called an extended tree of $t$. An example of an extended tree is shown in Figure 3.6. Notice that the inorder traversal of the extended tree of $t$ will be $m_1\alpha_1m_2\alpha_2\ldots\alpha_{n-1}m_n$ where $m_1m_2\ldots m_n$ are the inorder traversal of $t$ and $\alpha_1\alpha_2\ldots\alpha_{n-1}$ is the labeling of $t$. Observation 1 can be restated as follows:

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

![Figure 3.6 An example of an extended tree.](image)

### 3.2.2 Definition of twin binary sequences

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

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

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

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

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

**Lemma 1** For any binary tree, its labeling sequence $\alpha$ and directional bit sequence $\beta$ must satisfy conditions (1) and (2).

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

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

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

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

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

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

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

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

**Proof:** The one-to-one mapping between twin binary sequences and mosaic floorplan follows from Theorem 1 and the proof in paper [33] that the mapping between twin binary trees and mosaic floorplan is one-to-one.

### 3.2.3 From TBS to floorplan

#### 3.2.3.1 Algorithm for floorplan realization

In order to realize a floorplan from its TBS representation efficiently, we devised an algorithm that only needs to scan the sequences once from right to left to construct the packing. We will construct the floorplan by inserting the modules one after another following the \(\pi\) sequence in the reversed order. A simple example illustrating the step of the algorithm is given in Figure 3.7. At the beginning, we will put the last module of the...
sequence, i.e., module $D$, into the packing $P$. We will then insert the other modules one after another. The next module to be considered after $D$ is $\pi_4 = E$. Since $\alpha_4 = 0$, we will look at the sequence $\beta$ and find the closest bit "1" on the right of $\beta_4$, i.e., $\beta_5$. We will then add module $E$ into $P$ from the left pushing $D$ (since $\alpha_5 = D$) to the right as shown in Figure 3.7(b) and delete bit $\beta_5$ from $\beta$. The next module to be considered after $E$ is $\pi_3 = C$. Since $\alpha_3 = 1$, we will look at the sequence $\beta'$ and find the closest bit "1" on the right of $\beta'_3$, i.e., $\beta'_4$. We will then add module $C$ into $P$ from above pushing $E$ (since $\alpha_4 = E$) down as shown in Figure 3.7(c) and delete bit $\beta'_4$ from $\beta'$. These steps repeat until the whole sequence $\pi$ is processed and a complete floorplan is obtained.

Algorithm TBStoFloorplan

Input: A TBS $s = (\pi, \alpha, \beta, \beta')$

Output: Packing $P$ corresponding to $s$

Begin

1. Append $\alpha$ with bit '1', i.e., $\alpha_n = 1$.

2. Initially, we have only module $\pi_n$ in $P$.

3. For $i = n - 1$ down to $i = 1$:

4. If ($\alpha_i = 0$):

5. Find the smallest $k$ s.t. $i < k \leq n$ and $\beta_k = 1$.

6. Note that the set $S$ of modules $\pi_{i+1}, \pi_{i+2}, \ldots, \pi_k$ (those with corresponding $\beta$ bit not deleted yet) will be lying on the left boundary of $P$. Add module $\pi_i$ to $P$ from the left, pushing those modules in $S$ to the right.

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

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

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

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

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

End

Figure 3.7 A simple example of constructing a floorplan from its TBS.

3.2.3.2 Proof of correctness

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

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

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

Assume that the statement is true when $i \geq p$ for some $1 < p \leq n - 1$. Consider the case when $i = p - 1$. If $\alpha_{p-1} = 0$, similarly, $\pi_{p-1}$ must have a right child $\pi_j$ in $t_1$ according to the definition of TBS. Let $t$ be the subtree of $t_1$ rooted at $\pi_j$. Since we are performing the inorder traversal in the reversed order, the nodes in $t$ must have been scanned immediately before $\pi_{p-1}$. Let them be $\pi_p, \pi_{p+1}, \ldots, \pi_{p+m-1}$ where $m$ is the size of $t$. (Note that $p \leq j \leq p + m - 1$.) If there is any node $\pi_k$ in $t$ where $k \neq j$ and $\beta_k = 1$, $\beta_k$ must have been deleted when the scan reaches $\pi_{p-1}$. This is because if $\beta_k = 1$, $\pi_k$ is the right child of its parent $\pi_j$ in $t_1$ and $\pi_j$ must also be in $t$. According to the inductive hypothesis, when we scan to $\pi_j$, we will find that $\alpha_j = 0$ (since $\pi_j$ has a right child $\pi_k$ in $t_1$) and $\pi_k$ will be the only node in the right subtree of $\pi_j$ in $t_1$ such that $\beta_k = 1$ at that moment. Since the nodes in the right subtree of $\pi_j$ will be lying immediately in front of $\pi_j$ in the reversed inorder traversal, we will delete all the $\beta$ bits up to and including $\beta_k$. Therefore, when we scan to $\pi_{p-1}$, any node $\pi_k \in t$ where $k \neq j$ and $\beta_k = 1$ will have its $\beta$ bit deleted.

\begin{figure}[h]
\centering
\begin{subfigure}{0.2\textwidth}
  \centering
  \includegraphics[width=\textwidth]{figure-a.png}
  \caption{(a)}
\end{subfigure}
\hspace{0.5cm}
\begin{subfigure}{0.2\textwidth}
  \centering
  \includegraphics[width=\textwidth]{figure-b.png}
  \caption{(b)}
\end{subfigure}
\caption{Proof of Theorem 3.}
\end{figure}

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

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

Assume that the algorithm is correct when there are \( n \) modules in the floorplan for some \( n \geq 2 \). Consider the case when there are \( n + 1 \) modules. W.l.o.g., we assume that \( \alpha_1 = 0 \). The case when \( \alpha_1 = 1 \) can be proved similarly. Since \( \alpha_1 = 0 \), the upper left module \( A = \pi_1 \) has a right child \( B = \pi_j \) in \( t_1 \) and \( A \) should be packed in one of the two ways shown in Figure 3.8 in the floorplan \( F \). Assume that the TBS of \( F \) is \( s = (\pi, \alpha, \beta, \beta') \) where \( \pi = \pi_1, \pi_2, \ldots, \pi_{n+1} \), \( \alpha = \alpha_1, \alpha_2, \ldots, \alpha_n \), \( \beta = \beta_1, \beta_2, \ldots, \beta_{n+1} \) and \( \beta' = \beta'_1, \beta'_2, \ldots, \beta'_{n+1} \). Consider sliding module \( A \) out of the floorplan \( F \) (Figure 3.9) in the direction shown to obtain a floorplan \( F_1 \) with \( n \) modules. Note that the TBS \( s_1 \) for \( F_1 \) can be obtained from \( s \) by changing \( \beta_j \) from 1 to 0 and removing \( \pi_1, \alpha_1, \beta_1 \) and \( \beta'_1 \) from \( \pi, \alpha, \beta \) and \( \beta' \) respectively. Since \( F_1 \) has only \( n \) modules and the algorithm can construct the floorplan \( F_1 \) correctly from \( s_1 \) according to the inductive hypothesis.

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

The TBS representation is a complete and non-redundant representation for mosaic floorplan. Thus the number of different TBS configurations should give the Baxter number \([33]\). Baxter number can be written analytically as a complicated summation (equation 3.1 in \([33]\)). However, there is no known simple closed form expression for the Baxter number. In the following, an upper bound on the number of different TBS configurations (i.e., on the Baxter number) is presented.

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

3.3.1 Empty room in mosaic floorplan

A twin binary sequences \( s \) represent a mosaic floorplan \( F \). Now we want to insert an exact number of empty rooms at the right places in \( F \) to obtain a corresponding non-slicing floorplan \( F' \) such that every non-slicing floorplan can be generated by this method from one mosaic floorplan non-redundantly. There are two kinds of empty rooms. One is resulted because a big room is assigned to a small module. This kind of empty room is called \textit{reducible empty room}. An example is shown in Figure 3.10(a). Another kind of empty room is called \textit{irreducible empty room} and is defined as follows:

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

An example of an irreducible empty room is shown in Figure 3.10(b). We observed that an irreducible empty room must be of \textit{wheel} shape and its four \textit{adjacent rooms} (the rooms that share a T-junction at one of its corners) must not be irreducible empty rooms themselves:

![Figure 3.10 Examples of reducible and irreducible empty rooms.](image)

\textbf{Lemma 4} The T-junctions at the four corners of an irreducible empty room must form a \textit{wheel} structure (Figure 3.11).
Proof: If an empty room $X$ does not form a wheel structure, there is at least one slicing cut (Figure 3.12) on one of its four sides. By removing this slicing cut, we can merge $X$ with the room on the other side of the slicing cut (room A in Figure 3.12) and $X$ can be removed.

![Irreducible Empty Room](image)

**Figure 3.11** A wheel structure.

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

Proof: W.l.o.g., we consider an irreducible empty room $X$ of clockwise wheel shape, and assume that its adjacent room $A$ sharing with $X$ the T-junction at its upper left corner is also an irreducible empty room (Figure 3.13). Then $A$ must be an anti-clockwise wheel. There are two cases: (1) If $\text{width}(A) \geq \text{width}(S)$, $X$ can be merged with $A_1$ (Figure 3.13(a)) to form a new empty room. This empty room $X + A_1$ is reducible, and can be removed by combining with the modules on the right hand side (labeled $B$). (2) If $\text{width}(A) \leq \text{width}(S)$, $A$ can be merged with $X_1$ (Figure 3.13(b)) to form a new empty room and similar argument follows. In both cases, we are able to reduce the number of irreducible empty rooms by one. By repeating the above process, we will either end up with only one irreducible empty room that must satisfy the condition, or the situation that every remaining irreducible empty room does not share a T-junction with each other.
3.3.2 Mapping between mosaic floorplan and general non-slicing floorplan

In this section, we will show how a non-slicing floorplan $F'$ can be constructed from a mosaic floorplan $F$ by inserting some irreducible empty rooms at the right places in $F$. For simplicity, we will make use of twin binary trees for explanation. That means, given a mosaic floorplan $F$ represented by a twin binary trees $t_1$ and $t_2$, we want to insert the minimal number of empty rooms (represented by $X$) to the trees appropriately so that they will correspond to a valid non-slicing floorplan $F'$, and the method should be such that every non-slicing floorplan can be constructed by this method uniquely from one and only one mosaic floorplan. To construct a non-slicing floorplan from a mosaic floorplan, we only need to consider those irreducible empty rooms, because all reducible empty rooms can be removed by merging with some neighboring rooms. From Lemma 4, we know that an irreducible empty room must be of the shape of a wheel, so its structure in the twin binary trees must be of the form as shown in Figure 3.14. In our approach, we will use the following mapping $M_x$ to create irreducible empty rooms from a sliceline structure.
Definition 3  The mapping $M_x$ will map a vertical (horizontal) sliceline with one $T$-junction on each side to an irreducible empty room of anti-clockwise (clockwise) wheel shape (Figure 3.15).

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

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

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

From Lemma 5, we know that the adjacent rooms of an irreducible empty room must be occupied. Therefore if we want to insert $X$'s into the twin binary trees $t_1$ and $t_2$ of a mosaic floorplan, the $X$'s must be inserted between some module nodes as shown in Figure 3.16. Given this observation, we will first insert as many $X$'s as possible (i.e., $n - 1$) into $t_1$ and $t_2$ to obtain another pair of trees $t'_1$ and $t'_2$. An example is shown...
Figure 3.15  Mapping between mosaic floorplan and non-slicing floorplan.
in Figure 3.17(b). Now, the most difficult task is to select those X’s that are inserted correctly. According to Observation 2, a pair of twin binary trees are valid (correspond to a packing) if and only if the inorder traversal of their extended trees are equivalent except that all the bits are reversed. Therefore, in order to find out those valid X’s, we will write down the inorder traversal of the extended trees of $t'$ and $t''$ and try to match the X’s. The matching is not difficult since there must be an equal number of X’s between any two neighboring module names (Figure 3.17(c)). We may need to make a choice when there are more than one X’s between two modules. For example, in Figure 3.17(c), there is one X between C and D in the first sequence and there are two X’s in the second sequence. In this case, we can match one pair of X’s. There are two choices from the second sequence, and they will correspond to different non-slicing structures as shown in Figure 3.17(c). Every matching will correspond to a valid floorplan, and each non-slicing floorplan can be constructed uniquely by this method from one and only one mosaic floorplan.

3.3.3 Inserting empty rooms directly on TBS

In our implementation, we do not need to build the trees explicitly to insert empty rooms. We can scan the twin binary sequences $s = (\pi, \alpha, \beta, \beta')$ once to find out all the positions of the X’s in the inorder traversals of $t_1$ and $t_2$ after insertion. This is possible because of the following observation. Consider an X inserted at a node position A in a tree. If A has a left subtree B (Figure 3.16 (a)), this inserted X will appear just before the left subtree of A in the inorder traversal of $t'$. Similarly, if A has a right child

\begin{figure}[h]
\centering
\includegraphics[width=0.7\textwidth]{figure.png}
\caption{The only two ways to insert X into a tree.}
\end{figure}
A simple example of constructing a non-slicing floorplan from a mosaic floorplan.
B (Figure 3.16 (b)), this inserted X will appear just after the right subtree of A in the inorder traversal of t'. A simple algorithm can be used to break down the subtree structure of a tree and find out all the positions of the X's in the sequences after insertion in linear time. The details of the algorithm as follows.

We scan the TBS from left to right and assume that $\alpha_n = 1$. If $\alpha_t = 0$, module $\pi_t$ has a right subtree in $t_1$ according to the definition of TBS. By the observation above, we only need to find the position of the last module ($\pi_k$) in the right subtree of $\pi_t$ in $t_1$ from the TBS, and then insert one X just after $\pi_k$ in the inorder traversal of $t_1$. In addition, we will assign 1 as the labeling bit of the inserted X. Note that the right subtree of $\pi_t$ can be taken as a binary tree except that the directional bit of the root is 1, not 0 as usual. In addition, $\alpha_k = 1$. Thus, we obtain the modified conditions for the right subtree of $\pi_t$ as follows:

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

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

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

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

where we define

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

A simple example is shown in Figure 3.18. After we insert X for module $\pi_i$, the inorder traversal of the extended $t'_1$ becomes E1$\pi_i$0A0D1B0C1X1F0G. Note that the inserted
52

X for module \( \pi_i \) just appears after the last module (i.e., module \( C \)) of the right subtree of \( \pi_i \) in \( t_1 \). The labeling bit for the inserted \( X \) is 1.

<table>
<thead>
<tr>
<th>Tree ( t_1 )</th>
<th>TBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>( \pi ): E ( \pi ) A D B C F G</td>
</tr>
<tr>
<td>G</td>
<td>( \alpha ): 1 0 0 1 0 1 0 1</td>
</tr>
<tr>
<td>E</td>
<td>( \beta ): 0 0 1 0 1 1 0 1</td>
</tr>
</tbody>
</table>

Search the last module in the right subtree of \( \pi_i \) by counting the number of 1 and 0 in the bit sequence \( \beta \), \( \alpha \), \( \alpha \), ... until we reach module \( C \) which satisfies the equation:

\[
\sum_{j=1}^{l} (\beta_j + \alpha_j) = 2
\]

Figure 3.18 An example of searching the last module in the right subtree of \( \pi_i \).

If \( \alpha_i = 1 \), module \( \pi_i \) has a right subtree in \( t_2 \) according to the definition of TBS. Similarly, we can insert \( X \) for \( \pi_i \) directly into the inorder traversal of the extended \( t'_2 \) by searching the last module of the right subtree of \( \pi_i \) in \( t_2 \). The algorithm is exactly the same as above.

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

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

(b) For any proper prefix of the bit sequence \( \beta_{i-1} \alpha_{i-2} \beta_{i-2} \alpha_{i-3} \ldots \alpha_k \beta_k \alpha_{k-1} \), the number of 0's is less than or equal to the number of 1's plus 1.
Based on the above conditions, we can count the number of 0 and 1 from $\beta_{i-1}$ and $\alpha_{i-2}$ until we reach the module $\pi_k$. It is not difficult to find $\pi_k$ by the following mathematical form:

$$\sum_{j=i-1}^{k} (\tilde{\beta}_j + \tilde{\alpha}_{j-1}) = -2 \quad (3.2)$$

Another simple example is shown in Figure 3.19. After we insert $X$ for module $\pi_i$, the inorder traversal of the extended $t'_1$ becomes $G1F0X0A0D1B0C1\pi_0E$. Note that the inserted $X$ for module $\pi_i$ appears just before the first module (i.e., module $A$) of the left subtree of $\pi_i$ in $t_1$. The labeling bit for the inserted $X$ is 0.

If $\alpha_{i-1} = 0$, module $\pi_i$ has left subtree in $t_2$. Similarly, we can insert $X$ for $\pi_i$ directly into the inorder traversal of $t'_2$ by searching the first module in the left subtree of $\pi_i$ in $t_2$. The algorithm is exactly the same as above.

After we inserted all the possible $X$’s, we obtain the inorder traversals of the trees $t'_1$ and $t'_2$. Matching can then be done as described in the previous subsection.

Figure 3.19 shows an example of searching the first module in the left subtree of $\pi_i$.

### 3.3.4 Tight bound on the number of irreducible empty rooms

In order to describe non-slicing structure by a mosaic floorplan representation, some previous works [34, 35] include dummy blocks of zero area in the set of modules. The
method described in section 2.3 is very efficient but it is applicable to the TBS representation only. In general, we only need to have \( n - 1 \) extra dummy blocks in order to represent all non-slicing structures by a mosaic floorplan representation. We have proved an upper bound of \( n - 1 \) and a lower bound of \( n - 2\sqrt{n} + 1 \) on the number of irreducible empty rooms in a general non-slicing floorplan. (An example with 49 modules and 36 irreducible empty rooms is shown in Figure 3.20.) It means that \( n - 1 \) dummy blocks are needed and we cannot use much less.

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

**Proof:** According to Lemma 3, the adjacent rooms of an irreducible empty room in \( P \) must be occupied. Therefore, each irreducible empty room will take up four corners of some occupied rooms. Since there are only \( n \) occupied rooms in total and the four corners of the chip cannot be used, there are only \( 4n - 4 \) corners to be used. Therefore, there are at most \( n - 1 \) irreducible empty rooms. \( \square \)

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

**Proof:** A floorplan with \( n - 2\sqrt{n} + 1 \) irreducible empty rooms can be constructed similarly to the example in Figure 3.20. Let \( k \) be the number of modules along each edge (for the example in Figure 3.20, \( k = 7 \)), number of modules \( n = k^2 \) and number of empty rooms \( = (k - 1)^2 = (\sqrt{n} - 1)^2 = n - 2\sqrt{n} + 1 \). \( \square \)

### 3.4 Floorplan Optimization by Simulated Annealing

Simulated annealing is used to search for a good TBS. The temperature is set to \( 1.5 \times 10^6 \) initially and is lowered at a constant rate of 0.95 to 0.97 until it is below
Figure 3.20 Floorplan example with many irreducible empty rooms.

$1 \times 10^{-10}$. The number of iterations at one temperature step is 30. In every iteration of the annealing process, we will modify the TBS by one of the following four kinds of moves:

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

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

For move M1, we only exchange the module names in two randomly selected rooms. For move M2, we change the width and height of a module within the given limits of its aspect ratio. Obviously, both move M1 and M2 takes $O(1)$ time. For move M3 and
M4, we borrow and modify the idea of rotations in red-black tree [22]. A red-black tree is a binary search tree. The rotation in a red-black tree is an operation that changes the tree structure locally while preserving the inorder traversal of the tree. Two kinds of rotations, Right-Rotate and Left-Rotate, are defined originally in [22] (Figure 3.21). A and B represent two nodes. C, D and E represent arbitrary subtrees. Right-Rotate(T, A) transforms the left tree structure to the right tree structure, while keeping the inorder traversal of the tree unchanged (e.g., the inorder traversal of the tree before and after rotation are both equal to CBDAE in Figure 3.21). The operation of left rotation is similar. Both Left-Rotate and Right-Rotate run in \(O(1)\) time. When we apply red-black tree rotations on our twin binary trees, the subtree D in Figure 3.21 should not be 1 or 0. In the case that subtree D is 1 or 0, we modify the red-black rotations as shown in Figure 3.22, where D is designated to 0 or 1 after Right-Rotate(T, A) or Left-Rotate(T, B).

![Figure 3.21 Right-Rotate and Left-Rotate for a binary search tree](image)

![Figure 3.22 Modified red-black rotations when subtree D is 0 or 1](image)

For the moves M3 and M4, we randomly pick one module \(\pi_i\) from \(\pi\), and check \(\alpha_i\). If \(\alpha_i = 0\), \(\pi_i\) has a right child in \(t_1\) and \(\pi_{i+1}\) has a left child in \(t_2\). We can then use move
Figure 3.23  Four cases of $\text{Left-Rotate}(T, \pi_i)$ on $t$
M3 to apply \( \text{Left-Rotate}(T, \pi_i) \) on \( t_1 \) or use move M4 to apply \( \text{Right-Rotate}(T, \pi_{i+1}) \) on \( t_2 \). They are similar to each other and one of them will be randomly picked and applied. W.l.o.g., we present the details of \( \text{Left-Rotate}(T, \pi_i) \) on \( t_1 \) according to the following four cases shown in Figure 3.23(a), (b), (c) and (d). For simplicity, we use letter \( B, C \) and \( D \) to represent the root of each subtree.

**Case 1**: \( \beta_i = 0 \) and the right child of \( \pi_i \) has a left child.

**Case 2**: \( \beta_i = 1 \) and the right child of \( \pi_i \) has a left child.

**Case 3**: \( \beta_i = 0 \) and the right child of \( \pi_i \) has no left child.

**Case 4**: \( \beta_i = 1 \) and the right child of \( \pi_i \) has no left child.

For Case 1, after left rotation of module \( \pi_i \), the only change in \( t_1 \) is the directional bit of module \( A \) and \( C \), so we only need to flip \( \beta_A \) and \( \beta_C \). Because the labeling sequence \( \alpha \) does not change, we do not need to update \( t_2 \). Thus, we keep \( \beta' \) the same as before. Case 2 is similar to Case 1. For Case 3, both \( \alpha_i \) and the directional bit of module \( A \) are flipped after left rotation of module \( \pi_i \). In order to maintain conditions (1) and (2), we need to update \( t_2 \) by flipping one directional bit of \( \beta' \) from 0 to 1. Note that \( \pi_i \) is the left child of \( A \) in \( t_2 \). Thus, if \( \beta'_A \) is 0, we will flip \( \beta'_A \) from 0 to 1. Otherwise, we will flip \( \beta'_i \) from 0 to 1. Case 4 is similar to Case 3. Actually, updating \( t_2 \) in case 3 and 4 is exactly the \( \text{Right-Rotate}(T, A) \) on \( t_2 \) in case 3 and 4.

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

In move M3 and M4, if \( \alpha \) does not change, we only need to update one tree and each move takes \( O(1) \) time. If \( \alpha \) changes, we need to update both trees (i.e., apply two rotations). Therefore, both move M3 and M4 take \( O(1) \) time in practice.
Lemma 7  Starting from any TBS, we can generate any other TBS with the same $\pi$ sequence by applying one or more moves from the set \{M3, M4\}

Proof:  We observe that at most $n - 1$ Left Rotations suffice to transform any arbitrary $n$-node binary tree into a left-going chain [22]. Given a TBT, w.l.o.g., we can apply at most $n - 1$ Left Rotations by move M3. The binary tree $t_1$ will become a left-going chain (Figure 3.24(a)). Since move M3 always results in a TBT, the binary tree $t_2$ must also be transformed into a right-going chain (Figure 3.24(b)). The corresponding floorplan is shown in Figure 3.24(c).

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

![Figure 3.24 Proof of Lemma 7.](image)

3.5 Experimental Results

All experiments are carried out on a PC with 1400MHz Intel Xeon Processor and 256Mb Memory. Simulated annealing as stated in section 4 is used to search for a good
We test our algorithm using TBS with empty room insertion on six MCNC benchmarks. Besides, we also run the algorithm with empty room insertion disabled. In other words, only mosaic floorplan can be generated. For each case, two objective functions are considered. The first is to minimize area only. The second is to minimize a weighted sum of area and wirelength. The weights are set such that the costs of area and wirelength are approximately equal. Because of the stochastic nature of simulated annealing, for each experiment, ten runs are performed and the result of the best run is reported. The results for area minimization is listed in Table 3.1. The results for area and wirelength minimization is listed in Table 3.2.

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

In Table 3.3, we compare our results with ECBL [34] and the enhanced Q-sequences [35]. Notice that ECBL is run on Sun Sparc20 (248MHz) while Enhanced Q-seq is run on Sun Ultra60 (360MHz). We found that the scaling factors for the speeds of the three machines are 1:1.68:5.03. The runtimes reported in brackets in Table 3.3 are the scaled runtimes. We can see that the run time of TBS is much faster, although the performance of all three of them in area optimization are similar. We also compared TBS with those representations designed intrinsically for slicing structure. The performance of Fast-SP [31], Enhanced O-tree [29], B*-tree [21] and TCG [26] are shown in Table 3.4. Notice that Fast-SP and B*-tree are run on Sun Ultra1 (166MHz) while Enhanced O-tree and TCG are run on Sun Sparc20 (248MHz), and the scaling factors for their speeds are 0.613:1. Again, the runtimes reported in brackets in Table 3.4 are the scaled runtimes. We can see that TBS has again out-performed the other representations in terms of runtimes, while
Table 3.1 Area minimization.

<table>
<thead>
<tr>
<th>MCNC benchmark</th>
<th>TBS (with X)</th>
<th>TBS (no X)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% Dead-space</td>
<td>Run-time (s)</td>
</tr>
<tr>
<td>apte</td>
<td>1.89</td>
<td>0.86</td>
</tr>
<tr>
<td>xerox</td>
<td>2.17</td>
<td>1.30</td>
</tr>
<tr>
<td>hp</td>
<td>2.10</td>
<td>0.76</td>
</tr>
<tr>
<td>ami33a</td>
<td>3.05</td>
<td>1.26</td>
</tr>
<tr>
<td>ami49a</td>
<td>4.05</td>
<td>2.55</td>
</tr>
<tr>
<td>playout</td>
<td>6.20</td>
<td>2.58</td>
</tr>
</tbody>
</table>

Table 3.2 Area and wirelength minimization.

<table>
<thead>
<tr>
<th>MCNC benchmark</th>
<th>TBS (with X)</th>
<th>TBS (no X)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% Dead-space</td>
<td>Wire-length</td>
</tr>
<tr>
<td>apte</td>
<td>1.79</td>
<td>12652</td>
</tr>
<tr>
<td>xerox</td>
<td>2.64</td>
<td>14937</td>
</tr>
<tr>
<td>hp</td>
<td>1.32</td>
<td>4246</td>
</tr>
<tr>
<td>ami33a</td>
<td>8.41</td>
<td>6078</td>
</tr>
<tr>
<td>ami49a</td>
<td>9.40</td>
<td>29668</td>
</tr>
<tr>
<td>playout</td>
<td>5.19</td>
<td>2.373</td>
</tr>
</tbody>
</table>

the packing quality in terms of area is similar. TBS is thus a more desirable representation since its fast computation allows us to handle very large circuits and to embed more interconnect optimization issues in the floorplanning process.
Table 3.3  Comparisons with ECBL and enhanced Q-sequences.

<table>
<thead>
<tr>
<th>MCNC benchmark</th>
<th>Total area</th>
<th>ECBL [34]</th>
<th>Enhanced Q-seq [35]</th>
<th>TBS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Area</td>
<td>Runtime (s)</td>
<td>Area</td>
</tr>
<tr>
<td>apte</td>
<td>46.56</td>
<td>45.93 3</td>
<td>3 (3)</td>
<td>46.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>47.44</td>
</tr>
<tr>
<td>xerox</td>
<td>19.35</td>
<td>19.91 3</td>
<td>3 (3)</td>
<td>19.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19.78</td>
</tr>
<tr>
<td>hp</td>
<td>8.30</td>
<td>8.918 11</td>
<td>11 (11)</td>
<td>9.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.48</td>
</tr>
<tr>
<td>ami33</td>
<td>1.16</td>
<td>1.192 73</td>
<td>73 (73)</td>
<td>1.194</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.196</td>
</tr>
<tr>
<td>ami49</td>
<td>35.4</td>
<td>36.70 117</td>
<td>117 (117)</td>
<td>36.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36.89</td>
</tr>
</tbody>
</table>

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

Table 3.4  Comparisons with other representations for slicing floorplan.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Time (s)</td>
<td>Area</td>
<td>Time (s)</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>Time (s)</td>
<td>Area</td>
<td>Time (s)</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>Time (s)</td>
<td>Area</td>
<td>Time (s)</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>Time (s)</td>
<td>Area</td>
<td>Time (s)</td>
</tr>
<tr>
<td>apte</td>
<td>46.92</td>
<td>1 (0.61)</td>
<td>46.92</td>
<td>11 (11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>46.92</td>
<td>7 (4.29)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>46.92</td>
<td>1 (1)</td>
</tr>
<tr>
<td>xerox</td>
<td>19.80</td>
<td>14 (8.58)</td>
<td>20.21</td>
<td>38 (38)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19.83</td>
<td>25 (15.33)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19.83</td>
<td>18 (18)</td>
</tr>
<tr>
<td>hp</td>
<td>8.947</td>
<td>6 (3.68)</td>
<td>9.16</td>
<td>19 (19)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8.947</td>
<td>55 (33.72)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8.947</td>
<td>20 (20)</td>
</tr>
<tr>
<td>ami33</td>
<td>1.205</td>
<td>20 (12.26)</td>
<td>1.242</td>
<td>119 (119)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.27</td>
<td>3417 (2095)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.20</td>
<td>306 (306)</td>
</tr>
<tr>
<td>ami49</td>
<td>36.5</td>
<td>31 (19.00)</td>
<td>37.73</td>
<td>406 (406)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>36.8</td>
<td>4752 (2913)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>36.77</td>
<td>434 (434)</td>
</tr>
</tbody>
</table>

1 Using Sun Ultra60 machine  2 Using Sun Ultra60 machine
CHAPTER 4. CONGESTION DRIVEN FLOORPLANNING

4.1 Introduction

Floorplan design is to produce a chip-level plan of a set of circuit modules by determining their positions and shapes on the chip. It is the first stage of the physical design process. Hence, it has significant effects on overall circuit quality. Traditionally, in floorplanning stage, the major objective is to minimize area and total wirelength. The routability and congestion issues are not considered until global routing. However, due to the continued scaling of VLSI technology, the design of chip-level interconnect has become increasingly complicated [36]. Traditional floorplanners will produce floorplans with congested routing regions that are difficult to eliminate in later stages. Therefore, it is necessary to pay attention to the congestion optimization at floorplanning stage in order to realize the single-pass design methodology.

4.1.1 Previous work

In the past few years, several works have been proposed to address the congestion issue in floorplan design. Until recently, all previous congestion models in literature divide the whole chip area into tiles [37, 38, 39, 40, 41]. The number of wires crossing a tile boundary is estimated and is used as a measure of congestion. In other words, the underlying routing graph is a grid graph [36], in which each vertex corresponds to a tile and each edge corresponds to a tile boundary. There is a tradeoff between the accuracy of congestion estimation and the cost of computation. If the number of tiles is small, the
congestion estimation will be inaccurate. If the number of tiles is large, the computation will be expensive.

To estimate the congestion of each edge in the routing graph, previous approaches can be divided into two categories. The first category performs global routing on the grid graph [37]. Because the congestion estimation is performed inside the inner loop of the floorplanner, it is important to reduce the runtime of global routing. Thus, they restrict the routing geometry to L-shaped and Z-shaped. As a result, the congestion estimation may not correlate well with the real congestion. In addition, since all nets are routed one by one during global routing, even with restricted routing geometry, the computation is still very expensive.

The second category applies a probabilistic map to estimate the probability of a net crossing each boundary [38, 39, 40, 41]. The congestion of a boundary is the summation over all nets of the probability on that boundary. The previous publications differ mainly in their probabilistic maps for a net and in the way they handle blockages. This approach is more efficient than the restrictive global routing approach in the first category. Therefore, this idea is also commonly used in placement stage to estimate congestion [42, 43, 44, 45, 46]. Notice that the probability distribution of a net is determined independent of other nets. Whereas, a realistic global router routes a net based on current routing congestion information. Thus, the congestion estimated by the probabilistic approach can be very different from that by a global router. For example, in Figure 5.5, we have two 2-pin nets \{A, B\} and \{C, D\}. Their bounding boxes overlap in region \textit{II}. By the probability approach, we will reach the conclusion that routing region \textit{II} is more congested than regions \textit{I} and \textit{III}. However a global router can avoid congestion by routing net \{A, B\} in region \textit{I} and \{C, D\} in region \textit{III}. The resulting congestion in \textit{II} can even be less than that in \textit{I} and \textit{III}.

Recently, Lai et al. [47] proposed a novel approach which is very different from all previous approaches. For a floorplan of \(n\) modules, \(2n\) regions are defined according
to the structure of the floorplan. Each region contains several adjacent modules. The congestion for a region is evaluated as the wire density passing through the boundary of the region (i.e., number of wires connecting modules inside the region to those outside divided by the length of the region boundary). An $O(n \log n)$ time algorithm based on least common ancestor computation is presented to evaluate the congestion of all regions. This approach is very efficient. However, since most regions are quite large and only a single number is provided for each region, only coarse congestion information can be provided.

4.1.2 Our contributions

In this chapter, we present a new congestion estimation model which is efficient and accurate. The basic idea is to perform global routing by a flow based approach to minimize the maximum congestion over channel segments. A channel segment is a segment of a channel shared by two adjacent rooms in a floorplan. If we represent each room by a vertex and connect each pair of adjacent rooms by an edge, the resulting graph is called an inner dual graph [48]. (See Figure 4.2 for an example.) Note that each edge in the inner dual graph corresponds to one channel segment. We use the inner dual graph as the underlying topology in global routing. In floorplanning, the exact pin positions inside a module are still unknown. It is a waste of time to use a fine grid graph to estimate the
routing congestion. It is enough for global routing to list out the set of rooms that a net passes through without specifying its exact route inside each room. Therefore, the inner dual graph is an ideal choice as the underlying routing topology. The size of the inner dual graph is linear to the number of modules and is typically much smaller than that of grid graph. Hence, it is much more efficient to use.

Inner dual graph is an undirected graph. In order to avoid detour in the routing solution, for each set of nets originating from a particular module, we use a different routing graph by assigning different directions to the edges of the inner dual graph. In order to solve this problem, we interpret it as a flow problem and we relax the integral flow constraints. We design an efficient two-phase algorithm to tackle this fractional flow problem. In the first phase, we propose an Incoming Flow Balancing (IFB) technique to derive a good initial fractional routing solution. In the second phase, we present a Stepwise Flow Refinement (SFR) technique to iteratively reduce the maximum congestion of the solution in the first phase. We prove that SFR always converges to the optimal solution. Since we relax the integral flow constraints, the optimal solution by our algorithm will only be a lower bound on the maximum congestion. A valid global routing solution can be obtained by a simple rounding procedure. We show experimentally that the maximum congestion after rounding is only increased by 2.82% on average. It justifies the use of fractional flow to estimate the routing congestion.

We demonstrate our model by integrating it into a simulated annealing (SA) based floorplanner. The maximum congestion is used as part of the cost of SA. The experimental results show that, on average, our congestion-driven floorplanner can generate a much less congested floorplan (-36.44%) with a slight sacrifice in area (+1.30%) and wirelength (+2.64%). The runtime of the whole SA process is only increased moderately (+270%). The efficiency of our model is because of the use of inner dual graph, the simplicity of the two-phase algorithm, and the fact that we route a set of nets simultaneously rather than net by net.
The remainder of the chapter is organized as follows. In Section 2, we will give an overview of our congestion-driven floorplanner. In Section 3, we will present the two-phase algorithm used to solve the fractional flow problem in detail. In Section 4, a rounding procedure will be presented to derive a global routing solution from the fractional flow. The experimental results will be described in Section 5. Finally, the chapter will be concluded in Section 6.

4.2 Overview of Our Floorplanner

We make use of Twin Binary Sequences (TBS) [49] as our floorplan representation in simulated annealing. Basically, our congestion model can be employed with any floorplan representation. In our approach, we choose TBS representation because of two reasons. First, TBS itself is a very efficient and effective floorplan representation for mosaic floorplan and can be extended to represent general floorplan. Second, the inner dual graph of a floorplan can be easily obtained in TBS floorplan realization step.

In the annealing process, we use a 3-stage SA with three different cost functions in different temperature ranges to reduce runtime. Firstly, at high temperature range, we only consider area and total wirelength in the cost, i.e.,

\[
\text{Cost} = \text{Area} + \alpha \times \text{Wirelength}.
\]

Then, at medium temperature range, we add an accurate, although not optimal, maximum congestion \( M_{cong1} \) derived by only IFB as an additional part of the cost, i.e.,

\[
\text{Cost} = \text{Area} + \alpha \times \text{Wirelength} + \beta \times M_{cong1}.
\]

Finally, when the annealing process reaches low temperature range, we replace \( M_{cong1} \) with maximum congestion \( M_{cong2} \) derived by IFB and SFR. The cost thus becomes:

\[
\text{Cost} = \text{Area} + \alpha \times \text{Wirelength} + \beta \times M_{cong2}.
\]
The following section will describe how to estimate the congestion of a given floorplan in detail.

4.3 Congestion Estimation Model

In Section 3.1, we first introduce the inner dual graph and describe how to obtain the underlying routing graph from the inner dual graph. Then we illustrate the method of constructing the inner dual graph directly from TBS in Section 3.2. Based on the inner dual graph, in Section 3.3, we formulate the congestion minimization problem as a flow problem. Section 3.4 and 3.5 describe an efficient two-phase algorithm to tackle the problem formulated in Section 3.3.

4.3.1 Underlying routing graph

The exact pin positions inside each module is not given in the floorplanning stage. So it is not necessary to use a fine grid graph to estimate the routing congestion as in previous works. It is enough for global routing in floorplanning stage to list out the set of rooms that a net passes through without specifying its exact route inside each room.

Given a floorplan $R$, the room adjacency relationships can be described by channel segment which is defined as a segment of a channel shared by two adjacent rooms in the floorplan. For example, in Figure 4.2(a), the channel segment corresponding to the adjacent rooms $C$ and $D$ is highlighted. The room adjacency relationships can also be represented by the inner dual graph $G = (V, E)$ [48] where

$$V = \{v | v \text{ corresponds to a room of } R\}$$

$$E = \{\{u, v\} | u \text{ and } v \text{ are adjacent to each other in } R\}$$

See Figure 4.2(b) for an example. Note that there are one-to-one mappings between the rooms in $R$ and the vertices in $G$, and between the channel segments in $R$ and the edges
in $G$. In the rest of the chapter, the terms floorplan room and inner dual graph vertex, and channel segment and inner dual graph edge are used interchangeably.

The inner dual graph can be used as the underlying graph in global routing. The size of the inner dual graph is linear to the number of modules and is typically much smaller than the size of grid graph used in previous approaches. Hence, it is much more efficient to use. However, the inner dual graph is an undirected graph. If it is used directly as the underlying routing graph, the routing solution may have a lot of detour. We avoid detour by assigning directions to the edges of inner dual graph. Notice that different nets may require different direction assignments, but all nets originating from the same module share the same direction assignment. So for each set of nets originating from a specific module, we can derive a specific directed acyclic graph (DAG) $G'$ as the routing graph according to the following rule (as illustrated in Figure 4.3). Consider nets originating from a source room $s$, and consider a channel segment $e$ shared by a pair of adjacent rooms in floorplan $R$. By extending the channel segment, the floorplan region will be divided into two sides. We assign the direction of the edge $e$ in the inner dual graph to be from the room on the same side as the center of $s$ to the other side. See Figure 4.4 for an example. Notice that even with direction assignment, some detour may still occur. For example, in Figure 4.4, a net following the path $< D, E, C, A >$ may have detour.
depending on the exact pin positions inside rooms $D$ and $A$. However, with direction assignment, major detour can be avoided.

We observe that the underlying routing graph for a specific commodity obtained by the assignment above is a directed acyclic graph (DAG) in most cases. However, there exists a special case as illustrated in Figure 4.5. When $D$ is considered as a source room, a cyclic path $< A, B, C, E, A >$ exists in this graph. To obtain an acyclic underlying routing graph $G' = (V', E')$, we remove one edge from each cycle based on the rule described in Section 3.4.

### 4.3.2 Inner dual graph construction from TBS

The inner dual graph $G$ describes the neighborhood information between any two rectangular rooms. Given a floorplan in TBS, we can construct its inner dual graph in linear time by finding all pairs of adjacent rooms. In TBS representation, each floorplan is one-to-one mapped to a pair of twin binary trees $(t_1, t_2)$. $t_1$ and $t_2$ are obtained by
connecting, respectively, lower-left corners and upper-right corners of all rooms. We use the floorplan in Figure 4.6 as an example to illustrate how to obtain all room adjacency relationships as well as the length of each channel segment directly from TBS. We define a left-going (right-going) branch of a binary tree to be any right (left) child and all its left (right) descendants. For example, in $t_1$, the left-going branches are $\{C, B\}$ and $\{E, D\}$. In $t_2$, the right-going branches are $\{A\}$ and $\{F, C\}$. We notice that for each vertical channel (not including the boundaries), the room(s) on its right side corresponds to the room(s) in a particular left-going branch of $t_1$. The room(s) on its left side corresponds to the room(s) in a particular right-going branch of $t_2$. See the vertical channel highlighted in Figure 4.6 as an example. If there are $m$ vertical channels in the floorplan, there will also be $m$ left-going branches in $t_1$ and $m$ right-going branches in $t_2$. The branches have already been found in the original TBS packing procedure. Thus, in order to capture all horizontal adjacency relationships as well as the length of each vertical channel segment, we only need to compare the heights of rooms in a left-going branch of $t_1$ with those in a corresponding right-going branch of $t_2$ (i.e., $\{C, B\}$ with $\{A\}$, $\{E, D\}$ with $\{F, C\}$). Similarly, we can obtain the vertical adjacency relationships and horizontal channel segment lengths by considering the horizontal channels (i.e., the right-going branches in $t_1$ and the left-going branches in $t_2$).
4.3.3 Problem formulation

In our formulation, as in previous congestion estimation papers, we only handle 2-pin nets for the sake of simplicity. Notice that multi-pin nets can be easily broken down to several 2-pin nets by Minimum Spanning Tree or Rectilinear Steiner Tree techniques.

Our congestion model is meant to estimate the best maximum congestion over all possible global routing solutions. If we do not assign directions to the inner dual graph $G$, we can formulate the global routing problem as a flow problem with several commodities, where each commodity corresponds to a set of nets originating from a particular module.

We first introduce some notations.

- $N_i$: the set of neighboring vertices of vertex $i$.
- $c_i^k$: the demand of vertex $i$ for commodity $k$, i.e., the total number of nets with source vertex $k$ and sink vertex $i$.
- $f_{ij}^k$: the amount of flow from $i$ to $j$ for commodity $k$ for $\{i, j\} \in E$ and $j \neq k$.
- $\text{cap}_e$: the capacity of channel segment $e$ in $R$, i.e., the maximum number of nets that can cross it.
• $cong_e$: the congestion of channel segment $e$, i.e., the ratio of the number of nets crossing it to its capacity.

• $M_{cong}$: the maximum congestion over all channel segments of floorplan $R$.

Note that all flow amount $f_{ij}^k$ should be integral in order to be a valid global routing solution. Also note that $f_{ij}^k$ may be different from $f_{ji}^k$. For the capacity of channel segment, it is technology dependent. We can calculate it as follows. Let $l_e$ be the length of the channel segment $e$. Let $b$ be the sum of minimum wire width and minimum wire spacing. Then the routing capacity is calculated as $cap_e = \lfloor l_e/b \rfloor$. In general, the capacity can also be modified to model routing blockage.

The congestion of edge $e = \{i, j\}$ can be written as follows:

$$cong_e = \frac{\sum_k (f_{ij}^k + f_{ji}^k)}{cap_e}$$

Then, the flow problem can also be formulated as the following integer linear program (ILP):

Minimize $M_{cong}$

such that

$$\frac{\sum_k (f_{ij}^k + f_{ji}^k)}{cap_e} \leq M_{cong}, \forall e = \{i, j\} \in E$$

(4.1)

$$\sum_{j \in N_i} f_{ji}^k = \sum_{j \in N_i} f_{ij}^k + c_i^k, \forall i, k \in V \text{ s.t. } i \neq k$$

(4.2)

$$f_{ij}^k \geq 0, \forall i, j, k \in V$$

(4.3)

$$f_{ij}^k \in \mathbb{Z}, \forall i, j, k \in V$$

(4.4)

Constraint (4.2) is the flow conservation constraint. It specifies that for each commodity $k$ and for each vertex $i \neq k$, the total incoming flow equals the total outgoing flow plus the demand of vertex $i$. Note that by summing constraint (4.2) over all $i \neq k$, we can derive:

$$\sum_{j \in N_i} f_{kj}^k = \sum_{i \neq k} c_i^k, \forall k \in V$$
This means for commodity $k$, the total outgoing flow from vertex $k$ equals the total demand.

Since we restrict the flow direction for different commodity as described in Section 3.1, we need to add $O(n^2)$ constraints to the ILP formulation above. For each commodity $k$ and each edge $\{i, j\} \in E$, if the flow direction is from $i$ to $j$, we add the constraints $f_{ij}^k \geq 0$ and $f_{ji}^k = 0$.

ILP is known to be NP-complete. To tackle this problem, we first relax the integral flow constraint (4.4). Notice that the resulting problem is similar to the classical maximum concurrent flow problem [50]. However, in our problem, the flow direction on each edge may differ for different commodities. Our problem can be solved by any LP solver. However, it is too time consuming to be applied in the inner loop of the floorplanning process. Instead, we propose an efficient two-phase algorithm to derive the optimal fractional flow solution. The algorithm will be explained in detail in the following two subsections.

4.3.4 Incoming flow balancing (IFB) Phase

In this Section, we present an Incoming Flow Balancing (IFB) technique to derive a good fractional flow solution. This solution is included into the cost of the second stage of SA and is also used as an initial solution of SFR technique described in Section 3.5.

We construct the flow solution by iteratively deriving the flow of each commodity one by one based on current congestion information. At the beginning, we set the flow amount and congestion on each edge for each commodity to 0. When considering commodity $k$, we obtain the underlying routing graph $G' = (V', E')$ by assigning the directions to each edge of the inner dual graph $G$. If cycle occurs, we remove the most congested edge in the cycle according to the current congestion information. Then we consider vertices in reverse topological order$^{1}$ [51] of graph $G'$. For each vertex $i$, in order to minimize the

---

$^{1}$A reverse topological order of a directed acyclic graph (DAG) is a linear ordering of all its vertices such that if it contains an edge $(u, v)$, then $u$ appears after $v$ in the ordering. For instance, in Figure 4.4(b), the corresponding reverse topological order is $A, B, C, E, D$. 

maximum congestion, we balance incoming flow to make the congestion of incoming edges as even as possible. Let $d_{in}$ be the number of incoming edges for vertex $i$. Let $f_{in_j}$ be the flow amount of commodity $k$, $cong_{in_j}$ be the current congestion excluding commodity $k$, and $cap_{in_j}$ be the capacity of the $j$-th incoming edge ($1 \leq j \leq d_{in}$). Our goal to minimize the maximum congestion over all incoming flow can be written as follows:

$$\text{Minimize} \left\{ \max_{1 \leq j \leq d_{in}} \left( congr_{in_j} + \frac{f_{in_j}}{cap_{in_j}} \right) \right\}.$$ 

Since we consider the vertices in reverse topological order, all outgoing flow of $i$ has already been determined. Let $f_{out}$ be the total outgoing flow amount. Then the flow conservation constraint in equation (4.2) can be rewritten as follows:

$$\sum_{j=1}^{d_{in}} f_{in_j} = e^k_i + f_{out} \quad (4.5)$$

This problem can be easily solved by adding flow to the least congested incoming edge(s) until its congestion matches that of the next least congested edge. Note that there may be more than one edges with the least congestion. In that case, we add flow to them such that they still have the same congestion. We keep on adding flow until equation (4.5) is satisfied. See an example in Figure 4.7. For commodity $k$, The total outgoing flow amount is 6. The demand for vertex $i$ is 2. So the total incoming flow amount should be $6 + 2 = 8$. We assign $f_{in_1} = 0, f_{in_2} = 2, f_{in_3} = 6$ to edges 1, 2, and 3, respectively. The maximum incoming flow congestion is thus 0.6.

The procedure of routing all commodities once is called a pass. In IFB phase, we perform several passes until the maximum congestion converges. Notice that in pass $i \geq 2$, for commodity $k$, we first remove all its flow in pass $i - 1$ and update the congestion. Then we balance the flow of incoming edges according to the updated congestion. Since our algorithm is very greedy, the maximum congestion will converge in 2 to 4 passes in practice. An example of the convergence of the MCNC benchmark apte circuit is shown...
Demand: $C_{k} = 2$

\[
\begin{align*}
\text{congin}_1 &= 0.6 \\
\text{capin}_1 &= 10 \\
\text{congin}_2 &= 0.4 \\
\text{capin}_2 &= 20 \\
\text{congin}_3 &= 0.3 \\
\text{capin}_3 &= 30 \\
\end{align*}
\]

Min(Min(Min(congin1, congin2, congin3)) = 0.6

\[
\begin{align*}
\text{fin1} &= 0 \\
\text{congin}_1 &= 0.6 \\
\text{fin2} &= 2 \\
\text{congin}_2 &= 0.5 \\
\text{fin3} &= 6 \\
\text{congin}_3 &= 0.5 \\
\end{align*}
\]

Figure 4.7 Illustration of Incoming Flow Balancing technique. (a) Flow distribution before assigning incoming flow. (b) Flow distribution after assigning incoming flow.

\[
\begin{align*}
\text{Mcong} &= 0.514 \\
\text{Mcong} &= 0.528 \\
\text{Mcong} &= 0.876 \\
\end{align*}
\]

Figure 4.8 The convergence of maximum congestion in IFB phase for circuit apte.

in Figure 4.8. The overall flow of this phase is summarized in the IFB Algorithm below.

4.3.5 Stepwise flow refinement (SFR) phase

Since IFB phase can only achieve local incoming flow balance at each step, we still need an additional phase to obtain global solution by stepwise refining the flow solution given by IFB phase. An iteration in SFR phase is illustrated in Figure 4.9.

First, we pick an edge $e = \{i, j\}$ with maximum congestion $M_{cong}$. Second, we pick a commodity $k$ which contributes more than $\gamma\%$ of total flow amount on edge $e$ (i.e.,
IFB Algorithm:

**Input:** Inner dual graph $G = (V, E)$

**Output:** $cong_e$ and $f_{ij}^k$ on each vertex $k$ and edge $e = \{i, j\}$

Initialize $cong_e$ and $f_{ij}^k$ to 0 for all $k$ and $e = \{i, j\}$.

**While** $Mcong$ is not converged **do**

**For** each commodity $k$ **do**

- Assign flow directions to all edges to obtain DAG $G'$;
- If it is not the first pass,
  - remove $f_{ij}^k$, and update $cong_e$ for each edge;
  - Do a reverse topological ordering on DAG $G'$;
- **For** each vertex in a reverse topological order **do**
  - Assign incoming flow in a balanced manner;

Figure 4.9 SFR approach to globally optimize the maximum congestion. In this example, $e = \{i,j\}$ is with $Mcong$, $V_{out} = \{j,H,L,M,N\}$, $V_{in} = \{i,C,B,k\}$. After one iteration, we derive $r_1 = \langle B,D,E,M \rangle$ and $r_2 = \langle B,i,j,M \rangle$. 
Third, based on the DAG $G'$ of commodity $k$, we find the set of vertices $V_{out}$ which is reachable from $j$. $V_{out}$ includes $j$. Similarly, we find the set of vertices $V_{in}$ which can reach $i$. $V_{in}$ includes $i$. Then, by applying breadth first search (BFS), we find a simple path $r_1$ which links a vertex $p \in V_{in}$ and another vertex $q \in V_{out}$. In the meantime, we require that $r_1$ should not pass through edge $e$ and the maximum congestion over the edges of $r_1$ should be at least $\epsilon$ less than $M_{cong}$. At the same time, we derive another path $r_2$ which connects $p$ and $q$ by passing through edge $e$. Finally, we move $\delta f$ amount of flow from $r_2$ to $r_1$ in a way that the maximum congestion over the edges of $r_1$ and $r_2$ is minimized. $\delta f$ should also satisfy an additional constraint that the total incoming flow amount for each vertex on $r_2$ after moving $\delta f$ flow amount should not be less than its demand of commodity $k$. Finally, we update the congestion and flow amount on edges of $r_1$ and $r_2$. We keep applying the SFR technique on the edge with maximum congestion iteratively until it is not able to find $r_1$ and $r_2$ for the current most congested edge after all commodities are tried. In practice, we could speed up this process by tuning the parameter $\gamma$ and $\epsilon$. A convergence of $M_{cong}$ in SFR phase is shown in Figure 4.10, where the test circuit is apte, and $\gamma$ and $\epsilon$ are set to 10 and 0.001, respectively. We cannot find routes $r_1$ and $r_2$ to further improve $M_{cong}$ after 13 iterations. SRF Algorithm gives the overall flow of the procedure above.

**Lemma 8** If $\gamma = \epsilon = 0$, the SFR algorithm always converges to the optimal solution.

**Proof:** Note that if the current flow solution is not optimal, SFR can always find two routes $r_1$ and $r_2$ to reduce the maximum congestion. Since the maximum congestion is bounded below, the SFR algorithm always converges to the optimal solution.

### 4.4 Global Routing Solution Generation

Based upon the final floorplan solution at the end of simulated annealing, we obtain a global routing solution by a simple rounding technique applied to the fractional flow solu-
Figure 4.10  The convergence of $M_{cong}$ of $apse$ in SFR phase, where $\gamma = 10$, $\epsilon = 0.001$.

SFR Algorithm:

**Input:** Inner dual graph $G$ with initial solution obtained from IFB algorithm.

**Output:** Minimized maximum congestion $M_{cong}$.

**Do**

- Pick an edge $e = \{i, j\}$ with maximum congestion;
- For each $k$ with $f_{ij}^k$ (or $f_{ji}^k$) > $M_{cong} \cdot \text{cap}_e \cdot \gamma \%$
  - Find $V_{in}$ and $V_{out}$;
  - Find $r_1$ and $r_2$;
  - Move $\delta f$ from $r_2$ to $r_1$ and update the flow amount and congestion of edges on $r_1$ and $r_2$;

**While** edge $e$ can find $r_1$ and $r_2$
tion. For each commodity $k$, we round the incoming flow of vertices in reverse topological order. In the process, we maintain the flow conservation constraint in equation (4.2) by adjusting the flow of the least congested incoming edge. Once the rounding process for all commodities is finished, we update the congestion for each edge. Then, we apply SFR to optimize the global routing solution. It is important to stress that this time we only allow to move an integral amount of flow, namely $\delta f$, from one congested route to another less congested route.

Table 4.1 Experimental results for MCNC benchmarks.

<table>
<thead>
<tr>
<th>MCNC benchmark</th>
<th>Area ($\text{mm}^2$)</th>
<th>WL (mm)</th>
<th>$F_{Mcong}$ $\rightarrow$ $1M_{cong}$</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F1/F2</td>
<td>F1/F2</td>
<td>F1/F2</td>
<td>F1/F2</td>
</tr>
<tr>
<td>apte</td>
<td>48.227/47.898</td>
<td>132.67/134.33</td>
<td>0.607 $\rightarrow$ 0.611 0.371 $\rightarrow$ 0.373</td>
<td>2.10/4.69</td>
</tr>
<tr>
<td>xerox</td>
<td>20.243/19.864</td>
<td>147.38/152.01</td>
<td>1.059 $\rightarrow$ 1.070 0.938 $\rightarrow$ 0.945</td>
<td>2.58/4.47</td>
</tr>
<tr>
<td>hp</td>
<td>9.397/10.019</td>
<td>44.93/48.91</td>
<td>1.931 $\rightarrow$ 2.014 0.803 $\rightarrow$ 0.826</td>
<td>1.74/3.73</td>
</tr>
<tr>
<td>ami33a</td>
<td>12.468/12.636</td>
<td>64.88/64.91</td>
<td>1.453 $\rightarrow$ 1.502 0.980 $\rightarrow$ 0.993</td>
<td>4.21/13.63</td>
</tr>
<tr>
<td>ami49a</td>
<td>39.748/40.221</td>
<td>302.56/294.53</td>
<td>2.570 $\rightarrow$ 2.738 1.590 $\rightarrow$ 1.699</td>
<td>8.14/75.04</td>
</tr>
<tr>
<td>$(F2-F1)/F1$</td>
<td>+1.30%</td>
<td>+2.64%</td>
<td>-35.88% $\rightarrow$ -36.44%</td>
<td>+270%</td>
</tr>
</tbody>
</table>

4.5 Experimental Results

We implement the algorithm in the C programming language and test five MCNC benchmarks on a Sun4u machine with 8 GB memory and 750GHz Sparcv9 processor. The parameters of those benchmarks are listed in Table 4.2. The modules in the benchmark are soft modules with aspect ratio 0.5 to 2. To calculate the capacity of each channel segment, we assume the sum of minimum wire spacing and minimum wire width to be $6\lambda$. Since our congestion model is based upon 2-pin net, we decompose each multi-pin net into a set of 2-pin nets. In addition, we randomly choose one pin as the source since the benchmarks are lack of signal direction information. In the case that one pin is located
along the chip boundary, we assign this pin to its corresponding boundary room. Thus, each net starts from one room and ends at another room.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>modules</th>
<th>nets</th>
<th>2-pin nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>8</td>
<td>97</td>
<td>172</td>
</tr>
<tr>
<td>xerox</td>
<td>10</td>
<td>203</td>
<td>455</td>
</tr>
<tr>
<td>hp</td>
<td>11</td>
<td>83</td>
<td>226</td>
</tr>
<tr>
<td>ami33a</td>
<td>33</td>
<td>123</td>
<td>363</td>
</tr>
<tr>
<td>ami49a</td>
<td>49</td>
<td>408</td>
<td>545</td>
</tr>
</tbody>
</table>

In order to test the effectiveness and efficiency of our algorithm, we compare two floor-planners: F1, without congestion optimization; F2, with congestion optimization. In F1, a single stage simulated annealing is used to search for a floorplan aiming at minimizing total area and wirelength only. The maximum congestion is obtained by applying IFB and SFR to the final floorplan. In F2, we employ aforementioned 3-stage simulated annealing to obtain a floorplan with minimized weighted sum of area, wirelength, and maximum congestion. The weights are set such that the costs of area, wirelength, and congestion are approximately equal. The initial temperature in annealing process is set to $1.5 \times 10^6$ and drops down at a constant rate of 0.95 to 0.97 until it is below $1 \times 10^{-10}$. The number of iterations at one temperature step is 30. For each experiment, 10 runs are performed and the result of the best run is reported. The area, total wirelength, maximum congestion and total runtime are reported in Table 4.1. In terms of maximum congestion, $FM_{cong}$ denotes the maximum congestion based on fractional flow and $IM_{cong}$ denotes the maximum congestion based on integral flow after rounding. The experimental results show that the maximum congestion after applying rounding increases only by 2.82%, on average. This means our fractional flow based congestion model is fairly accurate in terms of estimation of the congestion for a floorplan.
From Table 4.1, we can notice that, compared to floorplanner F1, the congestion-driven floorplanner F2 can reduce the maximum congestion by 36.44\% on average. More specifically, for *ami33a*, *hp* and *xerox*, F1 is not able to produce routable final floorplan solutions (as their maximum congestions exceed 1), whereas, F2 is able to produce routable solutions. For *apte*, both F1 and F2 generate a routable floorplan solution. However, F2 can reduce maximum congestion by almost 40\% as compared to F1. For *ami49a*, both F1 and F2 are not able to produce a routable floorplan. Nevertheless, since the maximum congestion by F2 is much less than F1, the floorplan by F2 is more likely to be successfully routed if more detour is allowed. Figure 4.11 and Figure 4.12 show the floorplan obtained by F1 and F2, respectively. The thickness of the lines in the boundaries denote the degree of congestion. We can see significant difference in Figure 4.11 and Figure 4.12 for *ami49a* in terms of congestion distribution, while the packing area are about the same. Hence, F2 can reduce the overall congestion and improve routability of a circuit significantly in floorplanning stage with slight increase of area (+1.30\%), total wirelength (+2.64\%), and runtime (+270\%).

### 4.6 Conclusion and Discussion

In this chapter, we presented a flow based congestion estimation model for estimating the routing congestion in floorplanning level. This model is based on the inner dual...
The two-phase algorithm used in this model is optimal and efficient in estimating the maximum congestion of a floorplan. The experimental results show that the maximum congestion can be better optimized by incorporating this congestion model into floorplanner with slight sacrifice on area and wirelength. In the future, we plan to extend our algorithm to handle timing driven floorplanning and noise-aware floorplanning.
CHAPTER 5. CONGESTION ESTIMATION MODELS IN PLACEMENT

5.1 Introduction

The task in placement is to find a location for each cell such that all cells are completely contained in placement region and no two cells overlap with each other. Traditionally, in placement stage, the major objective is to minimize the total wirelength. But with growing complexity of chips (state-of-the-art chips have several million movable objects), not only wire length, but also routing congestion needs to be emphasized at the placement stage. A highly congested region in the placement often leads to routing detours around the region which results in a larger routed wire length and worse timing. Congested areas can also deteriorate the performance of global router and, in the worst case, create an unroutable placement in the fix-die regime [52]. Congestion is one of the main optimization objectives in global routing. However, the optimization performance is constrained because the cells are already fixed at this stage. Therefore, designer can save substantial time and resources by detecting and reducing congested regions during the planning stages. An efficient and yet accurate congestion estimation model is crucial to be included in the inner loop of floorplanning and placement design.

The congestion can be interpreted as a supply and demand problem for routing resources. The supply of routing resources can be roughly determined by technology parameters, such as die size, number of layers and position of preplaced macros. In planning stages, the demand of routing resources for each design solution can be predicted by
congestion estimation models.

In the past few years, several congestion estimation models have been proposed to be utilized in floorplanning and placement design. All existing congestion estimation models in placement divide the placement region into tiles [41, 53, 44, 43, 45, 56, 54, 58, 59]. These models estimate congestion based on either tile region [41, 53, 44, 43, 54] or tile boundary [45, 55, 56, 57, 58, 59]. For each tile or tile boundary, the expected number of wires routed within the tile or across tile boundary is compared to its capacity, i.e., the number of free routing tracks crossing the tile or tile boundary. Models defined for one can be easily modified for the other. Since there is no fundamental difference between these two approaches, we discuss them together. In this paper, congestion is estimated based on tile boundary.

Previous approaches can be divided into three categories. The first category performs global routing on tile graph [56, 54, 57, 55, 58]. However, the run time is quite expensive, especially when multiple iterations are required during optimization. The second category applies a probabilistic map to estimate the probability distribution on tile structure [43, 44, 53, 41]. The congestion of a tile (tile boundary) is the summation over all nets of the probability on that tile (tile boundary). The previous publications differ mainly in their probabilistic maps for a net and in the way they handle blockages. This approach is more efficient than the global routing based approach in the first category. Therefore, this idea is commonly integrated in placement stage to estimate and minimize congestion [57, 43, 44, 46]. The third category includes other approaches like [45, 59]. In [45], Rent’s Rule is used to estimate the peak congestion value and regional congestions on a chip and in [59], a normal distribution of the number of nets per tile is assumed in their congestion estimation model.

Most congestion estimation models in floorplanning are also tile based by either global routing approach [37] or probabilistic approach [38, 39, 40]. Recently, in [60] and [47], congestion is estimated based on rectangular dissection structure in a floorplan instead
of tile structure. These two models are much more efficient than previous ones used in floorplanning, but not applicable to placement.

Among all models discussed above, the most widely used is the probabilistic approach because of its simplicity and efficiency. However, the existing probabilistic approach mainly suffers from the following three drawbacks. First, the number of turns for each route is not limited in this approach. As a result, the predicted congestion could be quite different from the actual congestion by global router. Second, the probability distribution of a net is determined independent of other nets. That means the probability value assignment is not adaptive to the current congestion distribution. However, a global router routes a net based on current routing congestion distribution and also performs rip-up and re-route on congested regions. As a result, the probabilistic map could overestimate the congestion for heavily congested regions while underestimate the congestion for non-congested regions. Third, the existing probabilistic map is only efficient for the placement of small and medium circuits. While for large scale circuit, the run time of this approach is still quite expensive.

In this paper, three novel congestion estimation models are proposed to tackle these drawbacks. And we compare these models with LKS, a revision of probabilistic congestion estimation model proposed in [41] by experiments.

1. Simple Probabilistic Model (SPM) are proposed to avoid unlimited number of bends in a route. In addition, a linear time algorithm are applied in SPM to assign probability value. Experimental results show that SPM is faster and more accurate than LKS.

2. Multi-pin net Probabilistic Model (MPM) are also proposed to speed up SPM. Unlike SPM, MPM does not need to apply Minimum Spanning Tree (MST) algorithm to dissect multi-pin net into a set of two terminal nets. Instead, it directly applies SPM on each multi-pin net. As a result, MPM is about 21 times faster.
3. Post-Probability Processing (PPP) are proposed to modify the predicted congestion obtained by $SPM$ so that the probability assignment will be adaptive to the congestion distribution. The experimental results show that $PPP$ is quite effective to improve the quality of probabilistic map by $SPM$.

The remainder of the paper is organized as follows. In Section 2, we introduce and define some notations for congestion estimation based on tile structure in placement. Then, we summarize previous probabilistic model in [41] and propose its tile boundary version denoted as $LKS$. In Section 3, we present the details of Simplified Probabilistic Model ($SPM$) with its linear time probability assignment algorithm. In section 4, Multi-pin net Probabilistic Model ($MPM$) is proposed. In section 5, two steps of Post-Probability Processing ($PPP$) are presented in details. In Section 6, the experimental results show the comparisons of new models with $LKS$ in terms of their run time and correlation. Finally, the conclusion and discussion will be given in Section 7.

5.2 Notations

Given a placement solution with pin coordinate information of each net in net list, we discretize the placement region with a homogeneous rectangular mesh. We analyze the congestion for each tile boundary in the mesh. The number of tiles is a user-defined parameter which depends on the core area and process technology parameters of the placement.

Suppose the size of the mesh is $M \times N$, which means the rectangular mesh consists of $MN$ homogeneous tiles. An example of a $6 \times 7$ tile mesh is shown in Figure 5.1. We define several notations in this tile structure as follows.

- $T_{ij}$: tile with its coordinate $(i, j)$, where $1 \leq i \leq M$, $1 \leq j \leq N$. 

Figure 5.1 Tile structure for congestion estimation

- $RB_{ij}$: right boundary of $T_{ij}$.
- $UB_{ij}$: upper boundary of $T_{ij}$.
- $RC_{ij}$: the capacity of $RB_{ij}$, i.e., the number of available horizontal routing tracks crossing $RB_{ij}$.
- $UC_{ij}$: the capacity of $UB_{ij}$, i.e., the number of available vertical routing tracks crossing $UB_{ij}$.
- $RU^k_{ij}$: the probabilistic usage of $RB_{ij}$ for net $k$, i.e., the probabilistic amount of used horizontal routing tracks which cross $RB_{ij}$ for net $k$.
- $UU^k_{ij}$: the probabilistic usage of $UB_{ij}$ for net $k$, i.e., the probabilistic amount of used vertical routing tracks crossing $UB_{ij}$ for net $k$.
- $RD_{ij}$: demand on $RB_{ij}$, i.e., the total number of used horizontal routing tracks which cross $RB_{ij}$ for all nets.
- $UD_{ij}$: demand on $UB_{ij}$, i.e., the total number of used vertical routing tracks which cross $UB_{ij}$ for all nets.

Obviously, $RD_{ij}$ and $UD_{ij}$ can be derived by accumulating $RU^k_{ij}$ and $UU^k_{ij}$ over all nets, respectively. The horizontal (vertical) congestion on $RB_{ij}$ ($UB_{ij}$) is defined as the ratio of $RD_{ij}$ ($UD_{ij}$) on $RC_{ij}(UC_{ij})$. 
Before we explain our new models, we first go over Lou's model in [41] and propose a simplified version of this approach.

5.2.1 Lou's model and its revision (LKS)

Basically, Lou's model [41] is based on the supply and demand analysis of routing resources, where the supply is determined by technology parameters and the demand is computed by probabilistic congestion model. The model only handles two-pin net while multi-pin net needs to be broken into a set of two-pin nets first. For each two-pin net, this model considers all possible ways that a router can route. The number of turns within a possible route is not restricted as long as this route is monotonic. Then the model assigns the same probability to each possible route and computes the probabilistic track usages on each tile within the bounding box of the net.

More specifically, for a two-pin net $k$ covering an $m \times n$ mesh the model first computes $F(m, n)$ as the total number of possible ways to optimally route this net. $F(m, n)$ can be derived from Theorem 1.

*Theorem 1:*

- $F(m, 1) = F(1, n) = 1$.
- $F(m, n) = F(m - 1, n) + F(m, n - 1)$.

After that, the model computes the horizontal and vertical probabilistic usage, $P_x(i, j)$ and $P_y(i, j)$, on each tile $T_{ij}$ for net $k$. The expressions of $P_x(i, j)$ and $P_y(i, j)$ are listed
as follows, where $1 \leq i \leq m$ and $1 \leq j \leq n$.

$$P_x(i, j) = \frac{1}{F(m, n)} \times \begin{cases} F(m, n - 1) \quad i = 1, j = 1 \\ 1 \quad i = 1, j = n \\ F(m - i + 1, n - 1) \quad 1 < i < m, j = 1 \\ \frac{F(m, n - j + 1) + F(m, n - j)}{2} \quad i = 1, 1 < j < n \\ \frac{F(i, j)F(m-i+1, n-j) + F(i, j-1)F(m-i+1, n-j+1)}{2} \end{cases}$$

$$P_y(i, j) = \frac{1}{F(m, n)} \times \begin{cases} F(m - 1, n) \quad i = 1, j = 1 \\ 1 \quad i = 1, j = n \\ \frac{F(m+i-1, n) + F(m-i, n)}{2} \quad 1 < i < m, j = 1 \\ F(m - 1, n - j + 1) \quad i = 1, 1 < j < n \\ \frac{F(i, j)F(m-i, n-j+1) + F(i-1, j)F(m-i+1, n-j+1)}{2} \end{cases}$$

It should be noted that Lou’s model estimates congestion on tile region instead of tile boundary. We derive its tile boundary version, denoted as $LKS$, with the corresponding $RU_{ij}^k$ and $UU_{ij}^k$ in Theorem 2. Obviously, the usage assignments in $LKS$ are much simpler than those in Lou’s model.

**Theorem 2:**

$$RU_{ij}^k = \frac{F(i, j)F(m-i, n-j+1)}{F(m, n)} \quad 1 \leq i < m, 1 \leq j \leq n \quad (5.1)$$

$$UU_{ij}^k = \frac{F(i, j)F(m-i+1, n-j)}{F(m, n)} \quad 1 \leq i \leq m, 1 \leq j < n \quad (5.2)$$

**Proof:** Given net $k$ covering an $m \times n$ mesh as shown in Figure 5.2. Suppose the source $s$ and the drain $d$ are located at the center of tile in lower left and upper right corner, respectively. Then, the total number of possible routes which start from $s$ and arrive at $T_{ij}$ is $F(i, j)$ according to the definition of $F(i, j)$. The total number of possible routes which start from $T_{(i+1)j}$ and arrive at $d$ is $F(m-i, n-j+1)$. Thus for net $k$, the total number of possible routes which horizontally cross $RB_{ij}$ is $F(i, j)F(m-i, n-j+1)$. Since the total number of possible routes for net $k$ is $F(m, n)$, Eq. (5.1) holds. Similarly, we can prove Eq. (5.2).
Figure 5.2 Proof of theorem 1.

After we obtain $RU_{ij}^k$ and $UU_{ij}^k$ for each net $k$, we accumulate them over all nets to derive $RD_{ij}$ and $UD_{ij}$.

5.3 Simple Probabilistic Model (SPM)

Although LKS discussed above is already quite elegant, there are some drawbacks in this model.

Note that in LKS, the maximum number of bends on a possible route for a net covering an $m \times n$ mesh can be $\text{min}(m, n)$. In addition, it is assumed that a multi-bend route shares the same probability as $L$, $Z$ or $W$ shape route. However, a realistic router restricts the maximum number of bends on a route especially for global or semi-global nets. And most nets are routed in $L$, $Z$ or $W$ shape. Thus, the congestion estimation results by LKS can be quite different from actual result. In this subsection, a new probabilistic congestion model called Simple Probabilistic Model (SPM) is proposed as a remedy for LKS.

Unlike LKS, for a net $k$ covering an $m \times n$ mesh, SPM assumes every $RB_{ij}$ within the bounding box of net $k$ has the same horizontal probabilistic usage. And every $UB_{ij}$ within the bounding box of net $k$ also shares the same vertical probabilistic usage. $RU_{ij}^k$
and $UU_{ij}^k$ are given as follows.

\begin{align*}
    RU_{ij}^k &= \frac{1}{n} \quad 1 \leq i < m, 1 \leq j \leq n \quad (5.3) \\
    UU_{ij}^k &= \frac{1}{m} \quad 1 \leq i \leq m, 1 \leq j < n \quad (5.4)
\end{align*}

Note that the probabilistic usage assignment is equivalent to evenly placing $m$ vertical routing segments with weight of $1/m$ and $n$ horizontal routing segments with weight of $1/n$ within the bounding box of net $k$. The horizontal routing segments are illustrated in Figure 5.3. We denote these horizontal and vertical routing segments as weighted pseudo horizontal routes and vertical routes, respectively. These segments are called pseudo routes since they are not real in actual global routing. We only make use of them to predict the horizontal and vertical usages. Note that we have identical horizontal (vertical) probabilistic usage for each net, we thus propose an efficient algorithm to calculate $RD_{ij}$ and $UD_{ij}$.

W.L.O.G., we assume each weighted pseudo vertical route starts from bottommost tile and stops at uppermost tile while each weighted pseudo horizontal route starts from leftmost tile and stops at rightmost tile in the bounding box. Then for all nets, we define $RF_{ij}$ as the difference between the total amount of weighted pseudo horizontal routes which start from $T_{ij}$ and that which stop at $T_{ij}$. Similarly, $UF_{ij}$ is the difference between
the total amount of weighted pseudo vertical routes which start from \(T_{ij}\) and that which stop at \(T_{ij}\).

Note that for each net \(k\), we only need to update \(RF_{ij}\) for \(T_{ij}\) on leftmost and rightmost columns, and update \(UF_{ij}\) for \(T_{ij}\) on uppermost and bottommost rows in the bounding box. More specifically, suppose the coordinates of lower left tile in bounding box is \((x_1, y_1)\), and the upper right tile is \((x_2, y_2)\). We increase each \(RF_{x_1j} (y_1 \leq j \leq y_2)\) by \(1/n\) and decrease each \(RF_{x_2j} (y_1 \leq j \leq y_2)\) by \(1/n\). Similarly, we increase each \(UF_{iy_1} (x_1 \leq i \leq x_2)\) by \(1/m\) and decrease each \(UF_{iy_2} (x_1 \leq i \leq x_2)\) by \(1/m\). After we scan all nets one by one, \(RD_{ij}\) and \(UD_{ij}\) can be derived by following theorem.

**Theorem 3:**

\[
RD_{ij} = RF_{ij} \quad 1 \leq j \leq N \tag{5.5}
\]

\[
RD_{ij} = RD_{(i-1)j} + RF_{ij} \quad 1 < i < M, 1 \leq j \leq N \tag{5.6}
\]

\[
UD_{i1} = UF_{i1} \quad 1 \leq i \leq M \tag{5.7}
\]

\[
UD_{ij} = UD_{(j-1)j} + UF_{ij} \quad 1 \leq i \leq M, 1 < j < N \tag{5.8}
\]

**Proof:** Note that we assume every weighted pseudo horizontal route walks from left to right. For each \(RB_{ij} (1 \leq j \leq N)\), the total amount of routes which cross \(RB_{ij}\) equals the total amount of weighted horizontal pseudo routes which start from \(T_{ij}\). Since no horizontal pseudo route stops at \(T_{ij}\), the Eq.(5.5) in Theorem 3 thus holds. For each \(RB_{ij} (1 < i < M, 1 \leq j \leq N)\), the weighted pseudo horizontal routes which cross \(RB_{ij}\) consist of two kinds of routes. The first kind is the route which starts from \(T_{ij}\). The second kind is the route which crosses \(RB_{(i-1)j}\) while does not stop at \(T_{ij}\). By the definition of \(RD_{(i-1)j}\) and \(RF_{ij}\), the Eq. (5.6) holds. Similar proof can be done for Eq. (5.7) and (5.8).

An example to calculate horizontal usage is shown in Figure 5.4, where the graph only includes the weighted pseudo horizontal routes. And the weight of each pseudo horizontal route is assumed as 0.5.
The pseudocode of \textit{SPM} is summarized in Algorithm 1. We compare the time complexities for \textit{LKS} and \textit{SPM} as follows. Assume the number of multi-pin nets in a placement is $K$ and we use MST for multi-pin nets. Assume the size of mesh is $M \times M$ and the maximum number of pins for any net is $P$. Then both \textit{LKS} and \textit{SPM} take $O(P^2)$ \cite{61} to call MST for each net. Note that for each two-pin net $k$ covering an $m \times n$ mesh, \textit{LKS} needs to assign $2Mr - m - n$ probabilistic usages to tile boundaries (i.e., $(m-1)n$ times for $RU_{ij}$ and $(n-1)m$ times for $UU_{ij}$). While \textit{SPM} only needs to update $2f(m)n + 2f(n)m$ times of $RD_{ij}$ or $UD_{ij}$ (i.e., $2f(m)n$ times for $RD_{ij}$ and $2f(n)m$ times for $UD_{ij}$), where

$$f(x) = \begin{cases} 
0 & x = 1 \\
1 & x \geq 2 
\end{cases}$$ (5.9)

Thus the run time of \textit{LKS} is $O(KP^2 + M^2KP)$ and the run time of \textit{SPM} is $O(KP^2 + MKP + M^2)$ which equals $O(KP^2 + MKP)$.

In \textit{SPM}, we apply the same approach as described in Lou’s model to handle routing blockages (obstacles) in placement. We omit this part due to the page limitation.

### 5.4 Multi-pin Net Probabilistic Model (\textit{MPM})

Note that for each multi-pin net, \textit{LKS} and \textit{SPM} need to apply either Minimum Spanning Tree (MST) or Rectilinear Steiner Tree (RST) to convert it into a set of two-pin nets. However, the run time for MST takes a significant portion in the total run time.

\footnote{The MST algorithm we used is an $O(P^2)$ time implementation of Prim’s algorithm in RMST-Pack by Andrew Kahng and Ion Mandoiu downloaded from GSRC Bookshelf \cite{61}. This algorithm is significantly faster than $O(P\log P)$ algorithm \cite{?} due to small $P$ typically seen in VLSI circuits. The run time comparison between these two algorithms is referenced in \cite{61}}

\footnote{Lou’s model \cite{41} needs to assign $mn$ probabilistic usages to tiles.
Algorithm 1:
Initialize $RF_{ij}, UF_{ij}$ to 0
For each net in the design
  MST(net)
  For each segment of the MST
    Update $RF_{ij}$ and $UF_{ij}$ for the specified $T_{ij}$ within the bounding box.
  For each $RB_{ij}$
    Compute $RD_{ij}$ and horizontal congestion
  For each $UB_{ij}$
    Compute $UD_{ij}$ and vertical congestion.$\quad$

of LKS or SPM. This has been discussed in the previous subsection and will be verified in the experimental results.

Thus, for large scale design, the existing probabilistic congestion model is still quite expensive in terms of run time.

In order to speed up the estimation process, we propose an approach to directly apply SPM on multi-pin nets. The new model is called Multi-pin net Probabilistic Model (MPM). Instead of applying MST on multi-pin nets, MPM takes each multi-pin net as a weighted two-pin net. The bounding box of the weighted two-pin net is the rectangle which covers all pins of the multi-pin net. The weight $w(p)$ is a variable of the pin count, $p$, in a multi-pin net. $w(p)$ is listed in Table 5.1, which is cited from [43].

The experimental results show that MPM dramatically reduces the total run time for congestion estimation, especially for large size circuits. However, MPM is only a little bit worse than LKS in terms of quality.

5.5 Post-Probability Processing (PPP)

As we discussed before, existing congestion estimation models simply accumulate the probabilistic usage in tile or tile boundary for all nets. For the final congestion distribution, these nets are independent to each other. In other words, assignment of probabilistic
Table 5.1 Net weight $w(p)$ in MPM.

<table>
<thead>
<tr>
<th>pin count $p$</th>
<th>net weight $w(p)$</th>
<th>pin count $p$</th>
<th>net weight $w(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-3</td>
<td>1.0000</td>
<td>15-19</td>
<td>1.6899</td>
</tr>
<tr>
<td>4</td>
<td>1.0828</td>
<td>20-24</td>
<td>1.8924</td>
</tr>
<tr>
<td>5</td>
<td>1.1536</td>
<td>25-29</td>
<td>2.0743</td>
</tr>
<tr>
<td>6</td>
<td>1.2206</td>
<td>30-34</td>
<td>2.2334</td>
</tr>
<tr>
<td>7</td>
<td>1.2823</td>
<td>35-39</td>
<td>2.3895</td>
</tr>
<tr>
<td>8</td>
<td>1.3385</td>
<td>40-44</td>
<td>2.5356</td>
</tr>
<tr>
<td>9</td>
<td>1.3991</td>
<td>45-49</td>
<td>2.6625</td>
</tr>
<tr>
<td>10-14</td>
<td>1.4493</td>
<td>50-200</td>
<td>2.7933</td>
</tr>
</tbody>
</table>

Figure 5.5 Congestion estimation by existing probabilistic approach.

Usage for a specific net is not adaptive to current congestion distribution. As a result, the congestions on some boundaries are overestimated while others are underestimated. However, a global router typically tries to avoid over congested region by adaptively routing each net according to current congestion distribution or reduce over congested region as much as possible by performing post route optimization (or rip-up and reroute).

For example, in Figure 5.5, we have two 2-pin nets $\{A, B\}$ and $\{C, D\}$. Their bounding boxes overlap in region $II$. By existing probabilistic approach, we will reach the conclusion that routing region $II$ is more congested than regions $I$ and $III$. However a global router can avoid congestion by routing net $\{A, B\}$ in region $I$ and $\{C, D\}$ in region $III$. The resulting congestion in $II$ can even be less than that in $I$ and $III$. 
Post-Probability Processing (PPP) is a procedure proposed to deal with the over congested regions so that the estimated congestion correlates better with the actual result. Basically, this procedure can be applied to SPM, MPM or LKS. However, in our experiments, we only apply PPP on SPM since we want to achieve the best correlation in comparative time. Typically, PPP consists of two steps. Step one is to reassign the probabilistic usage to tile boundary for each net based on the congestion distribution obtained from SPM. Step two is to locally even out the congestion distribution of the over congested regions by moving certain amount of probabilistic usages from the over congested tile boundary to its neighbors.

These two steps are described in more details as follows.

In step one, we consider all nets one by one. Given a net covering an \( m \times n \) mesh, we have \( n \) pseudo horizontal routes and \( m \) pseudo vertical routes. For each pseudo horizontal route \( R_i \) (\( 1 \leq i \leq n \)), we find out its maximum congestion \( M_{C_i} \) over \( m - 1 \) \( R_{B_j} \) on \( R_i \). After that, from \( m \) pseudo horizontal routes, we pick two routes, denoted as \( A \) and \( B \), with largest \( M_{C_i} \) and smallest \( M_{C_i} \), respectively. Then we move the weight of route \( A \) to route \( B \), equivalently decrease each \( RD_{ij} \) on route \( A \) by \( 1/n \) and increase \( RD_{ij} \) on route \( B \) by \( 1/n \). A similar procedure will be performed on \( m \) pseudo vertical routes. An example of reassigning horizontal probabilistic usage is given in Figure 5.6. In this example, each \( RC_{ij} \) is 10 and the two-pin net covers a \( 4 \times 4 \) mesh. The horizontal usages before step one and after step one are depicted in Figure 5.6(a) and (b), respectively. 0.25 of horizontal usage is moved from pseudo horizontal route \( R_2 \) to \( R_4 \).

In step two, we first scan each \( RB_{ij} \) (\( 1 \leq i < M \) and \( 1 < j < N \)). If \( RD_{ij} \) exceeds \( RC_{ij} \), a global router will be supposed to perform rip-up and reroute on some nets which cross \( RB_{ij} \). In addition, we observe that most post route optimization procedure applies local detour to some nets in congested regions. In order to achieve more accurate congestion estimation on those over congested regions, we propose step two to emulate the effect of rip and reroute on congestion re-distribution. We locally even out the congestion dis-
tribution over the congested tile boundary with its six neighbors illustrated in Figure 5.7.
In practice, we replace these seven horizontal (or vertical) demands with their average demand, i.e., \( \text{AverDem} \), shown in the following equation.

\[
\text{AverDem} = \frac{(RD_{ij} + RD_{i(j-1)} + RD_{i(j+1)} + UD_{ij} + UD_{i(j-1)} + UD_{i(j+1)} + UD_{i+1j})}{7}
\]

A similar procedure is performed on vertical over congested tile boundaries. The experimental results show that \( PPP \) can significantly improve the quality of congestion estimation based on \( SPM \) while the run time is increased slightly.
5.6 Experimental Results

We implemented four models in C. The benchmarks in our experiments are derived from ISPD-02 suite [62] and tested on a Sun4u machine with 8 GB memory and 750MHz Sparcv9 processor. Statistics for placement benchmarks are given in Table 5.2. For each circuit, we place it using a wire-length driven placer, FastPlace [63], then perform routing by a global router which is based on maze routing and rip-up and reroute [64]. After that, we compare the congestion distributions generated by global router and four congestion estimation models. For each benchmark circuit, we take the number of rows as the size of the mesh. In order to test the effectiveness of our models on circuits with different levels of congestion, we provide two sets of capacities to the benchmark circuits: small capacities to test heavily congested circuits and large capacities to test lightly congested circuits. In addition, we define two metrics and measure the degree of correlation between the predicted and actual congestion.

- **Congestion Difference (CD):** the average on the difference between the predicted and actual routing congestion on tile boundaries. Since the congestion map is to detect the congested tile boundaries, we only count the tile boundaries with their actual routing congestions no less than 0.8.

- **Percentage of over Congested tile boundaries (PC):** the ratio of the number of tile boundary with its routing congestion larger than 1 over the number of tile boundaries in the design.

Note that a good correlation is indicated by a small $CD$ or a predicted $PC$ which is close to actual $PC$. 
Table 5.2 Placement Benchmark Statistics

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Nodes</th>
<th>#Nets</th>
<th>#Pins net</th>
<th>#Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>12506</td>
<td>14111</td>
<td>50566</td>
<td>96</td>
</tr>
<tr>
<td>ibm02</td>
<td>19342</td>
<td>19584</td>
<td>81199</td>
<td>109</td>
</tr>
<tr>
<td>ibm03</td>
<td>22853</td>
<td>27401</td>
<td>93573</td>
<td>121</td>
</tr>
<tr>
<td>ibm04</td>
<td>27220</td>
<td>31970</td>
<td>105859</td>
<td>136</td>
</tr>
<tr>
<td>ibm05</td>
<td>28146</td>
<td>28446</td>
<td>126308</td>
<td>139</td>
</tr>
<tr>
<td>ibm06</td>
<td>32332</td>
<td>34826</td>
<td>128182</td>
<td>126</td>
</tr>
<tr>
<td>ibm07</td>
<td>45639</td>
<td>48117</td>
<td>175639</td>
<td>166</td>
</tr>
<tr>
<td>ibm08</td>
<td>51023</td>
<td>50513</td>
<td>204890</td>
<td>170</td>
</tr>
<tr>
<td>ibm09</td>
<td>53110</td>
<td>60902</td>
<td>222088</td>
<td>183</td>
</tr>
<tr>
<td>ibm10</td>
<td>68685</td>
<td>75196</td>
<td>297576</td>
<td>234</td>
</tr>
<tr>
<td>ibm11</td>
<td>70152</td>
<td>81454</td>
<td>280786</td>
<td>208</td>
</tr>
<tr>
<td>ibm12</td>
<td>70439</td>
<td>77240</td>
<td>317760</td>
<td>242</td>
</tr>
<tr>
<td>ibm13</td>
<td>83709</td>
<td>99666</td>
<td>357075</td>
<td>224</td>
</tr>
<tr>
<td>ibm14</td>
<td>147088</td>
<td>152772</td>
<td>546816</td>
<td>305</td>
</tr>
<tr>
<td>ibm15</td>
<td>161187</td>
<td>186606</td>
<td>715823</td>
<td>303</td>
</tr>
<tr>
<td>ibm16</td>
<td>182980</td>
<td>190048</td>
<td>778823</td>
<td>347</td>
</tr>
<tr>
<td>ibm17</td>
<td>184752</td>
<td>189581</td>
<td>860036</td>
<td>379</td>
</tr>
<tr>
<td>ibm18</td>
<td>210341</td>
<td>201920</td>
<td>819697</td>
<td>361</td>
</tr>
</tbody>
</table>

5.6.1 Correlation

Table 5.3 and Table 5.4 report the congestion correlation between predicted results and actual results. Small and large routing capacities are used in Table 5.3 and 5.4, respectively. We assume $RC_{ij}$ is same as $UC_{ij}$. $PC_R$ represents $PC$ given by global router. Columns 3 to 6 give $CD$ of $LKS$, $MPM$, $SPM$ and $SPM_{PPP}$, respectively. Columns 8 to 11 give $PC - PC_R$ for $LKS$, $MPM$, $SPM$ and $SPM_{PPP}$, respectively. Obviously, a positive (negative) number means the corresponding model over (under) estimates congestion on this circuit. The best $CD$ and $PC - PC_R$ for each circuit in terms of congestion correlation are in boldface. Note that $Aver$ on $PC - PC_R$ is the average on the absolute value of $PC - PC_R$. We compare three proposed models with $LKS$ by their $Aver$.

Table 5.3 and 5.4 show that $SPM_{PPP}$ consistently has the best $CD$ among four
Figure 5.8 Congestion map by global router.

Figure 5.9 Congestion map by LKS.

Figure 5.10 Congestion map by MPM.
Figure 5.11  Congestion map by $SPM$.

Figure 5.12  Congestion map by $SPM_{PPP}$.
models. Compared with $CD_{LK}$, $CD_{SPM.PPP}$ is on average 21.80% and 19.90% smaller for highly congested and lightly congested circuits, respectively. $CD_{SPM}$ is on average 5.24% and 6.47% smaller than $CD_{LK}$ for highly congested and lightly congested circuits, respectively. While $CD_{MPM}$ is 8.06% and 9.95%, on average, larger than $CD_{LK}$ for highly congested and lightly congested circuits, respectively. In terms of $PC$ estimated by four models, $PC_{SPM.PPP}$ is closest to $PCR$, on average. Compared to $LK$, the average difference between $PC_{SPM.PPP}$ and $PCR$ is decreased by 46% and 90.79% for highly and lightly congested circuits, respectively. In other words, among these four models, $SPM.PPP$ shows the best correlation between its predicted congestion versus actual congestion.

We also notice that for all lightly congested circuits, both $LK$ and $SPM$ significantly overestimate the percentage of congested tile boundaries. In other words, $PPP$ can be taken as a remedy to overcome this drawback in $LK$ and $SPM$.

A congestion map visually plots the congestion in the design by assigning different colors to different congestion costs. A lighter color means higher congestion cost. From Figure 5.8 to 5.12, we plot the congestion maps of circuit ibm16 with both horizontal and vertical capacity of 17 by global router and four models.

5.6.2 Run time

Table 5.5 reports the run time for different models and global router. Due to the page limitation, we only report the run time for highly congested circuits. The run time for lightly congested circuits are very similar to that of highly congested circuits. In this table, Column 2 gives the run time used to call $MST$ on multi-pin nets. Columns 3, 4 and 6 give the run time used by $LK$, $SPM$, and $SPM.PPP$ to estimate congestion for two-pin nets given by $MST$. Column 5 gives the run time used to directly apply $MPM$ on nets given by circuits. If we do not count the run time of $MST$, $SPM$ is on average 2.05 times faster than $LK$. $LKS.PPP$ is on average 1.37 times faster than $LK$. If
the run time of \( MST \) is included, \( MPM \) is much faster than other three models due to the significant portion of run time used by \( MST \). Table 5.5 shows that \( MPM \) is on average 21.70 times faster than \( LKS \).

Overall, Compared to \( LKS \), \( SPM \) is slightly better in correlation and slightly faster in run time; \( MPM \) is comparative in correlation and much faster in run time; \( SPM_{PPP} \) is much better in correlation and slightly faster in run time.

5.7 Conclusion and Discussion

In this paper, we presented three different congestion estimation models for estimating the routing congestion in placement. All these models are based on probabilistic approach. An efficient yet more accurate probabilistic usage assignment is proposed in Simple Probabilistic Model (\( SPM \)). Multi-net Probabilistic Model (\( MPM \)) is proposed based on \( SPM \) to reduce the total run time for congestion estimation process. \( SPM_{PPP} \) is proposed to improve the correlation of \( SPM \). However, there is still a lot of room to improve the effectiveness of Post-Probability Processing. In the future, we plan to integrate our different congestion estimation model in different stage of congestion driven placement.
Table 5.3  Correlation of predicted congestion with actual results for heavily congested circuits.

<table>
<thead>
<tr>
<th>Ckt</th>
<th>Cap.</th>
<th>$CD$</th>
<th>$PC_R$</th>
<th>$PC - PC_R$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$LKS$</td>
<td>$MPM$</td>
<td>$SPM$</td>
</tr>
<tr>
<td>ibm01</td>
<td>9</td>
<td>0.240</td>
<td>0.322</td>
<td>0.215</td>
</tr>
<tr>
<td>ibm02</td>
<td>17</td>
<td>0.241</td>
<td>0.298</td>
<td>0.226</td>
</tr>
<tr>
<td>ibm03</td>
<td>18</td>
<td>0.258</td>
<td>0.267</td>
<td>0.225</td>
</tr>
<tr>
<td>ibm04</td>
<td>16</td>
<td>0.262</td>
<td>0.286</td>
<td>0.254</td>
</tr>
<tr>
<td>ibm05</td>
<td>28</td>
<td>0.189</td>
<td>0.194</td>
<td>0.176</td>
</tr>
<tr>
<td>ibm06</td>
<td>17</td>
<td>0.194</td>
<td>0.213</td>
<td>0.183</td>
</tr>
<tr>
<td>ibm07</td>
<td>17</td>
<td>0.258</td>
<td>0.289</td>
<td>0.249</td>
</tr>
<tr>
<td>ibm08</td>
<td>17</td>
<td>0.266</td>
<td>0.271</td>
<td>0.255</td>
</tr>
<tr>
<td>ibm09</td>
<td>16</td>
<td>0.245</td>
<td>0.262</td>
<td>0.233</td>
</tr>
<tr>
<td>ibm10</td>
<td>17</td>
<td>0.244</td>
<td>0.259</td>
<td>0.237</td>
</tr>
<tr>
<td>ibm11</td>
<td>17</td>
<td>0.250</td>
<td>0.260</td>
<td>0.237</td>
</tr>
<tr>
<td>ibm12</td>
<td>21</td>
<td>0.252</td>
<td>0.258</td>
<td>0.239</td>
</tr>
<tr>
<td>ibm13</td>
<td>19</td>
<td>0.261</td>
<td>0.289</td>
<td>0.249</td>
</tr>
<tr>
<td>ibm14</td>
<td>18</td>
<td>0.256</td>
<td>0.274</td>
<td>0.245</td>
</tr>
<tr>
<td>ibm15</td>
<td>22</td>
<td>0.282</td>
<td>0.288</td>
<td>0.267</td>
</tr>
<tr>
<td>ibm16</td>
<td>22</td>
<td>0.236</td>
<td>0.247</td>
<td>0.228</td>
</tr>
<tr>
<td>ibm17</td>
<td>22</td>
<td>0.249</td>
<td>0.259</td>
<td>0.241</td>
</tr>
<tr>
<td>ibm18</td>
<td>18</td>
<td>0.280</td>
<td>0.281</td>
<td>0.274</td>
</tr>
<tr>
<td>Aver</td>
<td></td>
<td>0.248</td>
<td>0.268</td>
<td>0.235</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Decr</td>
<td>-8.06%</td>
<td>5.24%</td>
<td>21.80%</td>
<td>10%</td>
<td>0%</td>
<td>46%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Average on abstract value of $PC - PC_R$
Table 5.4 Correlation of predicted congestion with actual results for lightly congested circuits.

<table>
<thead>
<tr>
<th>Ckt</th>
<th>Cap.</th>
<th>( CD )</th>
<th>( PC_R )</th>
<th>( PC - PC_R )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LKS MPM SPM PPP</td>
<td>LKS MPM SPM PPP</td>
<td></td>
</tr>
<tr>
<td>ibm01</td>
<td>12</td>
<td>0.227 0.277 0.207 0.165</td>
<td>1.9</td>
<td>8.8 5.6 8.7 0.9</td>
</tr>
<tr>
<td>ibm02</td>
<td>22</td>
<td>0.221 0.279 0.208 0.165</td>
<td>0.8</td>
<td>8.9 9.7 8.9 -0.1</td>
</tr>
<tr>
<td>ibm03</td>
<td>23</td>
<td>0.179 0.187 0.160 0.133</td>
<td>0.1</td>
<td>8.5 6.3 8.2 -0.1</td>
</tr>
<tr>
<td>ibm04</td>
<td>22</td>
<td>0.211 0.233 0.200 0.159</td>
<td>0.5</td>
<td>6.4 5.2 6.6 0.3</td>
</tr>
<tr>
<td>ibm05</td>
<td>32</td>
<td>0.133 0.140 0.121 0.098</td>
<td>0.0</td>
<td>11.1 14.8 10.7 0</td>
</tr>
<tr>
<td>ibm06</td>
<td>20</td>
<td>0.149 0.174 0.135 0.115</td>
<td>0.0</td>
<td>9.0 5.7 8.9 0</td>
</tr>
<tr>
<td>ibm07</td>
<td>22</td>
<td>0.276 0.311 0.269 0.240</td>
<td>2.5</td>
<td>5.1 4.6 5.1 -1.4</td>
</tr>
<tr>
<td>ibm08</td>
<td>22</td>
<td>0.283 0.281 0.272 0.242</td>
<td>2.8</td>
<td>3.6 3.9 3.7 -0.6</td>
</tr>
<tr>
<td>ibm09</td>
<td>20</td>
<td>0.168 0.190 0.149 0.129</td>
<td>0.0</td>
<td>7.8 6.3 7.4 0</td>
</tr>
<tr>
<td>ibm10</td>
<td>21</td>
<td>0.188 0.201 0.173 0.142</td>
<td>0.2</td>
<td>8.0 7.3 8.1 0.2</td>
</tr>
<tr>
<td>ibm11</td>
<td>21</td>
<td>0.211 0.223 0.197 0.161</td>
<td>0.1</td>
<td>9.9 7.8 9.5 0.3</td>
</tr>
<tr>
<td>ibm12</td>
<td>24</td>
<td>0.250 0.251 0.238 0.201</td>
<td>3.6</td>
<td>7.1 6.5 7.0 -2.1</td>
</tr>
<tr>
<td>ibm13</td>
<td>24</td>
<td>0.192 0.219 0.179 0.158</td>
<td>0.0</td>
<td>8.0 7.5 7.8 0</td>
</tr>
<tr>
<td>ibm14</td>
<td>23</td>
<td>0.192 0.205 0.178 0.150</td>
<td>0.2</td>
<td>7.3 7.1 7.3 0.1</td>
</tr>
<tr>
<td>ibm15</td>
<td>32</td>
<td>0.199 0.207 0.187 0.157</td>
<td>0.0</td>
<td>7.6 8.0 7.1 2.0</td>
</tr>
<tr>
<td>ibm16</td>
<td>26</td>
<td>0.154 0.174 0.144 0.128</td>
<td>0.1</td>
<td>5.3 6.0 5.1 0.6</td>
</tr>
<tr>
<td>ibm17</td>
<td>28</td>
<td>0.191 0.204 0.181 0.176</td>
<td>0.1</td>
<td>7.6 8.3 7.4 2.0</td>
</tr>
<tr>
<td>ibm18</td>
<td>25</td>
<td>0.196 0.226 0.192 0.179</td>
<td>0.1</td>
<td>5.9 7.2 5.9 2.3</td>
</tr>
<tr>
<td>Aver</td>
<td></td>
<td>0.201 0.221 0.188 0.161</td>
<td>7.6* 7.1* 7.4* 0.7*</td>
<td></td>
</tr>
</tbody>
</table>

| Decre. | -9.95% 6.47% 19.90% | 6.58% 2.63% 90.79% |

* Average on abstract value of \( PC - PC_R \)
Table 5.5  Run time for congestion estimation models and global router.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>(MST(s))</th>
<th>(LKS(s))</th>
<th>(SPM(s))</th>
<th>(MPM(s))</th>
<th>(SPM_{PPP}(s))</th>
<th>(Router(h))</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>2.479</td>
<td>0.332</td>
<td>0.240</td>
<td>0.142</td>
<td>0.302</td>
<td>1.150</td>
</tr>
<tr>
<td>ibm02</td>
<td>5.469</td>
<td>0.584</td>
<td>0.416</td>
<td>0.203</td>
<td>0.513</td>
<td>3.267</td>
</tr>
<tr>
<td>ibm03</td>
<td>4.148</td>
<td>0.908</td>
<td>0.438</td>
<td>0.281</td>
<td>0.594</td>
<td>7.417</td>
</tr>
<tr>
<td>ibm04</td>
<td>6.798</td>
<td>0.999</td>
<td>0.503</td>
<td>0.322</td>
<td>0.714</td>
<td>7.917</td>
</tr>
<tr>
<td>ibm05</td>
<td>7.758</td>
<td>1.773</td>
<td>0.658</td>
<td>0.285</td>
<td>1.226</td>
<td>14.617</td>
</tr>
<tr>
<td>ibm06</td>
<td>6.112</td>
<td>1.054</td>
<td>0.628</td>
<td>0.344</td>
<td>0.782</td>
<td>8.417</td>
</tr>
<tr>
<td>ibm07</td>
<td>6.992</td>
<td>1.710</td>
<td>0.851</td>
<td>0.459</td>
<td>1.379</td>
<td>27.300</td>
</tr>
<tr>
<td>ibm08</td>
<td>11.470</td>
<td>2.103</td>
<td>1.195</td>
<td>0.472</td>
<td>1.663</td>
<td>21.517</td>
</tr>
<tr>
<td>ibm09</td>
<td>8.971</td>
<td>2.286</td>
<td>1.293</td>
<td>0.563</td>
<td>1.779</td>
<td>31.617</td>
</tr>
<tr>
<td>ibm10</td>
<td>11.733</td>
<td>4.921</td>
<td>1.846</td>
<td>0.730</td>
<td>3.055</td>
<td>63.417</td>
</tr>
<tr>
<td>ibm11</td>
<td>10.757</td>
<td>3.106</td>
<td>1.568</td>
<td>0.758</td>
<td>2.454</td>
<td>46.450</td>
</tr>
<tr>
<td>ibm12</td>
<td>13.325</td>
<td>5.565</td>
<td>2.106</td>
<td>0.758</td>
<td>3.680</td>
<td>58.617</td>
</tr>
<tr>
<td>ibm13</td>
<td>13.715</td>
<td>4.329</td>
<td>2.185</td>
<td>1.050</td>
<td>3.320</td>
<td>41.050</td>
</tr>
<tr>
<td>ibm14</td>
<td>18.613</td>
<td>9.240</td>
<td>3.654</td>
<td>1.617</td>
<td>5.941</td>
<td>97.467</td>
</tr>
<tr>
<td>ibm15</td>
<td>26.349</td>
<td>11.367</td>
<td>5.126</td>
<td>2.064</td>
<td>7.649</td>
<td>103.667</td>
</tr>
<tr>
<td>ibm16</td>
<td>26.814</td>
<td>10.896</td>
<td>5.123</td>
<td>2.044</td>
<td>8.181</td>
<td>156.400</td>
</tr>
<tr>
<td>ibm17</td>
<td>34.918</td>
<td>17.684</td>
<td>6.263</td>
<td>2.355</td>
<td>10.944</td>
<td>195.350</td>
</tr>
<tr>
<td>ibm18</td>
<td>30.367</td>
<td>11.012</td>
<td>5.523</td>
<td>2.385</td>
<td>8.583</td>
<td>212.967</td>
</tr>
</tbody>
</table>

**Speedup**  | \(2.05^* (1.14^+)\) | 21.70 | \(1.37^* (1.07^+)\)

* Average times faster than \(LKS\) without considering run time of \(MST\). + Average times faster than \(LKS\) with run time of \(MST\) included.
CHAPTER 6. EFFICIENT RECTILINEAR STEINER TREE CONSTRUCTION WITH RECTILINEAR BLOCKAGES

6.1 Introduction

Given \( n \) points on a plane, a Rectilinear Steiner Minimal Tree (RSMT) connects these points through some extra points called steiner points to achieve a tree with minimal total wire length. Many works have been done on this fundamental problem in VLSI physical design. However, most of them did not take blockages into consideration. In fact, today’s design often contains many rectilinear routing blockages, e.g., macro cells, IP blocks, and pre-routed nets. Thus, rectilinear Steiner minimal tree construction with rectilinear blockages (RSMTRB) becomes a very practical problem.

Generally, RSMT is used in initial net topology creation for global routing or incremental net tree topology creation in physical synthesis. And it is also utilized to accurately estimate congestion and wire length in early design stages, like block floorplanning and cell placement. The timing and congestion information obtained from RSMT can be used as a criteria in timing and congestion driven routing. It is the problem applied hundreds of thousands times and many of them have very large input sizes, RSMT thus deserves much intensive research in VLSI CAD.

Unfortunately, RSMT itself was shown to be strongly NP-complete by Shi [65]. Taking blockages into account dramatically increases the problem complexity. Thus, it is extremely unlikely that an efficient optimal algorithm exits for RSMTRB. Although there exist some heuristic algorithms for this problem, they have either poor performances
Existing heuristics for $RSMTRB$ can be classified into three categories.

The first category is maze routing [66] based approach. Maze routing can optimally routes two pin nets. However, for multi-pin nets, designers need to introduce a multi-terminal variant, which incurs a solution far from optimal. In addition, since the runtime and memory used in maze routing are proportional to the size of the routing area rather than the size of actual problem (i.e., the number of pins and blockages). Maze routing algorithms are inefficient in runtime and memory.

The second category is called sequential approach which typically consists of two steps. Step 1 is to construct a tree $T_1$, which is either a minimum spanning tree (MST) or a Steiner minimal tree (SMT) with absence of blockages. Step 2 is to transform $T_1$ to a $RSMT$ with blockages by substituting edges around the blockages for the edges overlapped by the blockages. These approaches are commonly used in industry due to its simplicity and efficiency. However, since step 1 neglects the global view of blockages, step 2 can only locally remove overlap between $T_1$ and blockages. The quality (i.e., the total wire length) of resulting $RSMT$ can be much worse than expected in many cases. A commonly used approach is illustrated in Figure 6.1. This approach is to first construct an $MST$ without considering any blockages. Then a $RSMT$ without overlapping with any blockage is constructed through a simple line sweep technique.

Later on, Yang et. al [67] introduce a complicated 4-process heuristics to remove the overlaps in step 2 in a more wise way. However the approach still can not avoid bad solution for many cases. This is because a bad $T_1$ due to neglecting blockage in step 1 could introduce an unexpected routing detour in step 2.

The third category consists of connection graph [68] based approaches. Typically, the approach in this category is to first construct a connection graph by pins and blockage boundaries, which guarantees that at least one rectilinear Steiner minimal (or close to minimal) tree is embedded in the graph. Then, some graph searching technique is used to
find a RMST as a subgraph from the connection graph. Unlike sequential approach, this approach can globally catch the view of both pins and blockages in the design. Therefore, the connection graph based approach can generally achieve an optimal (or near optimal) RMST.

The efficiency of connection graph based approach depends on the size of graph. While the accuracy or effectiveness of this approach depends on whether the graph contains a good steiner minimal tree. Obviously, there is a trade-off between efficiency and accuracy in this approach.

In [69], a connection graph is called escape graph which is constructed by escape segments. The number of vertex in the escape graph can be $O(n^2)$ in the worst case, where $n$ is the sum of pins and blockage boundaries. Even the size of a reduced escape graph is still quite huge. In [70], authors proposed a connection graph with $O(n \log n)$ vertices and edges and later on [71], they introduced an even smaller graph which contains $O(n^{3/2} \sqrt{\log n})$ vertices and $O(n \log^{3/2} n)$ edges. The construction of the connection graph takes $O(n \log^{3/2} n)$ time and memory usage.

In this chapter, we will propose an efficient and effective connection graph. It is called spanning graph. We show that the spanning graph contains only $O(n)$ vertices and $O(n)$ edges, which is smaller than any previous connection graph. In addition, we show that
our spanning graph can always produce a RMST with good quality. Due to the special property of the spanning graph, the construction takes only \(O(n \log n)\) time and memory usage, which are also smaller than those in any previous connection graph construction.

We organize the rest of the chapter as follows. In section 6.2, we will formally define the problem and explain the details of the basic component in the problem. In section 6.3, we will first demonstrate the drawback in the escape graph and introduce the spanning graph as a connection graph in RSMTRB. Section 6.4 describes the details of construction of spanning graph for RMSTRB. The experimental results are shown in section 6.5. The chapter is concluded in section 6.6

### 6.2 Problem Formulation

Let \(P = \{p_1, p_2, p_3, \ldots, p_m\}\) be a set of pins for \(m\) pin net. Let \(B = \{b_1, b_2, b_3, \ldots, b_k\}\) be a set of rectangular blockages. Let \(V = \{v_1, v_2, v_3, \ldots, v_n\} = P \cup \{\text{corners in } B\}\) as the vertex set in the problem, where each \(v_i\) has coordinates \((x_i, y_i)\). Note that each rectangular blockage has 4 corners, we have \(n \leq m + 4k\). The rectilinear distance between \(v_i\) and \(v_j\) is given as \(|x_i - x_j| + |y_i - y_j|\). A RSMT connects all pins through some extra points (called Steiner points) to achieve a minimal total length, while avoiding the intersection with any blockage in the design.

#### 6.2.1 Rectilinear blockages

If all boundaries of a blockage are either horizontal or vertical, we call this blockage as rectilinear blockage. Note that each rectilinear blockage can be dissected into a set of rectangular blockages (see Figure 6.2 as an example). In the rest of chapter, for simplicity, we assume each blockage to be rectangular.
6.2.2 Directional blockages

In multi-layer routing, there exist three types of blockages. The first one is called complete blockage which blocks all vertical and horizontal metal layers in its obstructed area. A complete blockage requires all routes must detour around it. The second type is denoted as horizontal blockage in which all vertical layers in obstructed area are blocked while a certain number of horizontal layers are still available for routing. The routes are allowed to horizontally pass through the obstructed area, while not allowed in vertical direction. The third type is called vertical blockage, where routes can still horizontally pass through the obstructed area. These three types of directional blockages are illustrated in Figure 6.3.
6.3 Escape Graph Versus Spanning Graph

6.3.1 Redundancy in escape graph

As we mentioned in Section 6.1, a connection graph can catch the global view of both blockages and pins. And the efficiency of this approach highly depends on the size of the connection graph. In other words, a good connection graph is able to describe all necessary geometrical relationship between pins and blockages using as few edges as possible. In [69], the connection graph is called escape graph, which is constructed by escape segments. Escape segments are formed by horizontal and vertical lines extending from pins and blockage boundaries, and ending with their abutment to either a blockage boundary or the internal perimeter of the routing region. The number of vertex in the escape graph can be $O(n^2)$ in the worst case. An example is shown in Figure 6.4. The collection of the escape segments (shown as dashed segments) composes the escape graph. And the graph preserves a good RSMT for a multi-pin net. However, we notice that most of edges and vertices are redundant in the escape graph to find a RSMT. For example, instead of using 13 edges, 3 edges (shown as solid segments) are enough to represent the connection relationship between the blockage $b_2$ and pin $p_2$ in the corresponding connection graph.
6.3.2 Spanning graph

In [72], a spanning graph is introduced as an intermediate step in minimal spanning tree construction. Given a set of points on the plane, a spanning graph is an undirected graph over the points that contains at least one minimal spanning tree. The number of edges in the graph is called the cardinality of the graph.

The construction of spanning graph is illustrated in Figure 6.5. From each point \( p \), a plane can be divided into 8 regions by horizontal, vertical and \( \pm 45^\circ \) lines through \( p \). It can be proved that the rectilinear distance between any two points in one region is always smaller than the maximal distance from them to \( p \). Due to the cycle property of a minimal spanning tree, that is, the longest edge on any cycle should not be included in any minimal spanning tree, which means only the closest point to \( p \) in each region needs to be connected to \( p \). Considering all given points, the connections will form a spanning graph of cardinality \( O(n) \). In other words, spanning graph is able to describe the relative geometrical relationship between points in the plane using \( O(n) \) edges.

Enlightened by the spanning graph in MST construction, we borrow this idea to solve RSMTRB. The details are presented in following section.
6.4 Spanning Graph Based Approach in RSMTRB

6.4.1 Search regions

In general spanning graph, each point \( p \) corresponds to 8 regions which are divided by the horizontal, vertical and \( \pm 45^\circ \) lines going through \( p \). Then we search each region and find the closest point in each region and connect it to \( p \). While in RSMTRB, for each blockage \( b_i \), we divide the whole plane into 8 regions as shown in Figure 6.6(a). Each corner of blockage \( b_i \) has three neighboring search regions which are adjacent to \( p_j \). We call these three regions as neighboring search regions for a given corner \( c \). For example, the neighboring search regions for the lower left corner \( p_1 \) in Figure 6.6(a) are \( R_8, R_1 \) and \( R_2 \). For each pin \( p \), we divide the whole plane into four search regions. The corresponding neighboring search regions for \( p \) are \( R_1, R_2, R_3 \) and \( R_4 \) as shown in Figure 6.6(b).

For each \( v \in V \), we connect the closest visible point in each neighboring search region to \( p \). Note that the visible point in each search region could be different for complete blockage and directional blockage. For example, the only difference between Figure 6.7(a) and (b) is that the blockage \( b_3 \) is a complete blockage in (a) while a horizontal blockage in (b). The search region \( R_2 \) for blockage \( b_4 \) is denoted as the enclosed area in the dashed segments. Obviously, the search region in (b) is larger than that in (a). In Figure 6.7(a), there exists only one visible point in region \( R_2 \) of \( b_4 \). While in Figure 6.7(b), there are three visible points in \( R_2 \) of \( b_4 \).

The reason why we only consider three search region for each corner \( p \) of blockage \( b \) is that for any visible point \( q \) in the rest five search regions of \( b \), we can always find another corner \( p' \) of \( b \) such that \( q \) lies in one search region of \( p' \) and a shortest path from \( p \) to \( q \) can be obtained by making \( p' \) as an intermediate point in the path.

In addition, the reason why we only search the visible points is that between the visible region and invisible region, there must exist at least one intermediate blockage, thus the point \( p \) can always reach the points in inviable region by making use of the boundaries.
of the intermediate blockage(s) as part of the shortest path. For example in Figure 6.8, region $R_2$ of blockage $b_2$ is invisible to the upperleft corner $p$ of $b_1$. Note that any point in $R_2$ of $b_2$ can be reached from $p$ by using boundaries of $b_2$ as part of the shortest path.

6.4.2 Spanning graph construction in RSMTRB

First, from blockage set $B$, we find out a subset $B_s$ which contains each blockage lying within or intersected by the bounding box of net. Let $V_s$ be a point set which includes all
corner of $B_s$ and pins. Our initial spanning graph $G$ has $V_s$ as its vertices and all blockage boundary segments of $B_s$ as its edges. Then, we incrementally construct the graph by applying a sweep line based edge connection among vertices in $V_s$. For any point $v \in V_s$, we only connect at most one visible point $v'$ in each neighboring search region to $v$, where $v' \in V_s$ and $v'$ is the closest point to $v$ in the corresponding neighboring search region.

Based on an $O(n \log n)$ sweep line algorithm proposed in [72], we propose a revised sweep line algorithm to construct spanning graph in $RSMTRB$ as follows. Our construction consists of four passes, each pass performs edge connection for a pair of search regions for $V_s$. For sake of simplifying the exposition, we will only present the detail of pass 1 which performs edge connection for search region $R_2$ and $R_6$ of corners. The rest of regions are similar to $R_2$ and the discussion can be easily extended to handle them.

First, all points in $V_s$ are sorted by their $x$ coordinates in non-decreasing order. Note that for each blockage, lowerleft corner shares the same $x$ coordinate as upperleft corner. Similarly, lowerright corner shares the same $x$ coordinate as upperright corner. Thus, we can sort left boundary and right boundary segment of each blockage instead of point by point to speed up sorting process. Note that a fundamental operation of sweep line algorithm is to keep an active set $A$ of if such that all points in $A$ are visible to $v$. We thus build an active set $A$ and dynamically keep $A$ by adding and deleting points for set $A$. We start with an empty set $A$ and check each point $v$ in the sorted list $V_{ss}$. If $v$ is not a lowerleft corner of a blockage (i.e., $v$ is a pin or any of other three corners in a blockage), we just add $v$ into set $A$, otherwise we perform edge connection as follows. Suppose $v$ is the lowerleft corner of blockage $b_i$. We first pick the point $v'$ which is right after $v$ in the list of $V_{ss}$. Note that $v'$ must be the upperleft corner point of $b_i$ due to the property of our sorting. Then, we check each point $a$ in $A$. If point $a$ lies in region $R_2$ of $b_i$, we add this point to another set $A_s$ and if $b_i$ is not a horizontal blockage, we delete this point from $A$. After that, we find out two points $q$ and $q'$ from $A_s$ which are closest to $v$ and $v'$, respectively, in rectilinear distance. Finally, we add two edges, $(v, q)$ and $(v', q')$, into
graph $G$. Note that $q$ and $q'$ could be the same point in $A_s$. Before we move to the next point in $V$, we vocate set $A_s$.

An example is shown in Figure 6.9. The bold segments are the edges added to graph $G$ after edge connection is performed for $R_2$ of corners. Since search region $R_6$ has the reverse sweep sequence, we can make use of the sorted list $V^{ss}$ to perform edge connection for $R_6$ in Pass 1. Similarly, in pass 2, we perform edge connection for search region $R_4$ and $R_8$ of blockages after sorting $V^{ss}$ in the non-decreasing order with their $y$ coordinates. In pass 3, we perform similar edge connection for $R_1$ and $R_5$ of blockage and $R_1$ and $R_3$ of pin after sorting $V^{ss}$ in non-decreasing $x + y$. Similarly, in pass 4, edge connection is performed for search region $R_3$ and $R_7$ of blockage and $R_2$ and $R_4$ of pin after $V^{ss}$ is sorted in non-decreasing $y - x$.

To achieve $O(n)$ running time, the active sets $A$ and $A_s$ must be efficiently maintained so that searching, deletion, and insertion each can be done in $O(\log n)$ time. The spanning graph after four passes is shown in Figure 6.10. The algorithm of spanning graph construction in RSMTRB is summarized in Algorithm 1. The details of edge connection for search region $R_2$ is summarized in Algorithm 2.

6.5 RMST Construction

After we complete the spanning graph, we apply a heuristic to construct an RMST based on the graph $G$. The heuristic consists of six steps.

Step 1: Construct the complete undirected graph $G_1 = (V_1, E_1)$ from $G$ and $P$. in such a way that $V_1 = P$ and for every $v_i, v_j \in E_1$, the length on the edge $v_i, v_j$ is equal to the length of the shortest path from $v_i$ to $v_j$.

Step 2: Find the minimum spanning tree $T_1$ of $G_1$. If there exist several minimum spanning trees, pick an arbitrary one.

Step 3: Construct the subgraph $G_s$ of $G$ by replacing each edge in $T_1$ by its corre-
Figure 6.9  Edge connection for region $R_2$.

Figure 6.10  Complete spanning graph
Algorithm 1: Spanning graph construction in RSMTRB

Input: $V_s$

1. sort $V_s$ by non-decreasing $x$;
2. perform edge connection for $R_2$ and $R_6$ of corners;
3. sort $V_s$ by non-decreasing $y$;
4. perform edge connection for $R_4$ and $R_8$ of corners;
5. sort $V_s$ by non-decreasing $x + y$;
6. perform edge connection for $R_1$ and $R_5$ of corners and $R_3$ and $R_7$ of pins;
7. sort $V_s$ by non-decreasing $y - x$;
8. perform edge connection for $R_3$ and $R_7$ of corners and $R_2$ and $R_4$ of pins;

Return: spanning graph $G$ for a net

Algorithm 2: edge connection for $R_2$

Input: a sorted $V_{ss}$ with non-decreasing $x$.

$A = \phi$;

For each $v \in V_{ss}$ {
    if ($v$ is a pin, lowerright or upright corner) {
        add $v$ to $A$;
    }
    else if ($v$ is a lowerleft corner of $b_i$) {
        if ($A! = \phi$) {
            $A_s = \phi$;
            if ($b_i$ is not a horizontal blockage)
                delete points from $A$ which are located in $R_2$ of $v$;
                add the points located in $R_2$ of $v$ to $A_s$;
            if ($A_s! = \phi$) {
                find point $q$ and $q'$ from $A_s$ which are closest to $v$ and $v'$, respectively;
                add new edge $(v, q)$ and $(v', q')$ to graph $G$;
            }
        }
    }
}
spending shortest path in $G$. If there are several shortest paths, pick an arbitrary one.

Step 4. Find minimum spanning tree $T_s$ of $G_s$. If there are several minimum spanning trees, pick an arbitrary one.

Step 5. Construct a Steiner tree $T_h$ from $T_s$ by deleting edges in $T_s$, if necessary, so that all the leaves in $T_h$ are pins.

Step 6. Rectilinearize $T_h$ to obtain a rectilinear steiner tree $T_r$. The above six steps are illustrated by an example in Figure 6.11.

6.6 Experimental Results

We implemented the spanning graph based $RMSTRB$ algorithm in C++ language. We compile and run the program on Intel Pentium 4 machine with 2.80GHz frequency and 1.5GB RAM. We run 3 industrial testcases and compare our approach with the traditional sequential approach illustrated in Figure 6.1. The statistic data of testcases are listed in Table 6.1. The total wirelength and runtime comparison is given in Table 6.2.

The experimental results show that our spanning graph based approach can reduce $14.31\%$ (on average) wirelength of $RMST$, comparing to sequential approach. And the runtime only increases $47.40\%$, on average.

Table 6.1 Statistics of testcases

<table>
<thead>
<tr>
<th>Test cases</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of instances</td>
<td>437444</td>
<td>277356</td>
<td>450367</td>
</tr>
<tr>
<td>number of I/O pins</td>
<td>774</td>
<td>1453</td>
<td>1276</td>
</tr>
<tr>
<td>number of nets</td>
<td>477380</td>
<td>285556</td>
<td>451250</td>
</tr>
<tr>
<td>2 pin net (%)</td>
<td>56.3</td>
<td>70.3</td>
<td>75.0</td>
</tr>
<tr>
<td>3-10 pin net (%)</td>
<td>21.2</td>
<td>16.3</td>
<td>16.8</td>
</tr>
<tr>
<td>10-50 pin net (%)</td>
<td>17.6</td>
<td>10.1</td>
<td>6.3</td>
</tr>
<tr>
<td>50-100 pin net(%)</td>
<td>5.1</td>
<td>3.5</td>
<td>1.6</td>
</tr>
<tr>
<td>≥ 100 pin net(%)</td>
<td>0.1</td>
<td>0.8</td>
<td>0.3</td>
</tr>
<tr>
<td>number of blockages</td>
<td>136</td>
<td>539</td>
<td>487</td>
</tr>
</tbody>
</table>
Figure 6.11  Six steps to construct RSMT
Table 6.2 Experimental results.

<table>
<thead>
<tr>
<th>Testcase</th>
<th>Total wirelength (μm)</th>
<th>Total runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential</td>
<td>Ours</td>
</tr>
<tr>
<td>1</td>
<td>2854122</td>
<td>2400499</td>
</tr>
<tr>
<td>2</td>
<td>2452867</td>
<td>2051254</td>
</tr>
<tr>
<td>3</td>
<td>1022723</td>
<td>913584</td>
</tr>
<tr>
<td>On average</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.7 Conclusion and Discussion

In this chapter, we propose an efficient and effective approach to construct rectilinear steiner minimum tree with rectilinear blockages. The connection graph we used in this approach is called spanning graph which only contains $O(n)$ edges and vertices. An $O(n \log n)$ time algorithm is proposed to construct spanning graph for RSMTRB. The experimental results shows that this approach can achieve a solution with significantly reduced wirelength. The total runtime increased is negligible in the whole design flow.

A possible extension of this approach is to perform congestion and timing global routing. We can handle the routing congestion by modifying the cost of each edge in spanning graph and deal with timing by controlling search direction in graph searching.
BIBLIOGRAPHY


[71] K. L. Clarkson, S. Kapoor and P. M. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n \log^{3/2} n)$ time. Unpublished manuscript.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this dissertation. First of all, I would like to thank my God for His bountiful grace to give me strength and wisdom to finish this work. I am also greatly indebted to my thesis advisor Professor Chris Chu for introducing me to the area of Physical Design. It is to his constant motivation and encouragement that I owe this work. I really appreciate his guidance and support throughout this research and the writing of this dissertation.

I would also like to thank my committee members for their efforts and contributions to this work: Dr. Randall Geiger, Dr. Degang Chen, Dr. Akhilesh Tyagi and Dr. Lu Ruan.

I thank all my previous members: Sampath Dechu, Nataraj Viswanathan, Rejesh and Min Pan in VLSI CAD group of ECE department of Iowa State University.

I also thank my co-workers in Cadence for their rich input and useful feedback for this dissertation, especially thank Ying Meng Li, John Shu, Robert Tien for their contribution to this dissertation.

Last, while not least, I would like to thank my parents, my parents-in-law, my dear wife, Jane and baby Samuel, for their strong support during the past months.