TCG-BASED PLACEMENT DESIGN HANDLING COMPLEX TOPOLOGICAL CONSTRAINTS FOR INTEGRATED CIRCUIT LAYOUTS

RUI HE

TCG-Based Placement Design Handling Complex Topological Constraints for Integrated Circuit Layouts

by

© Rui He

Master of Engineering

A thesis submitted to the
School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Engineering

Faculty of Engineering and Applied Science
Memorial University of Newfoundland

September 2011

St. John's Newfoundland Canada

Abstract

In modern VLSI design, extensive research has shown that automated analog layout generation is a nontrivial process in the analog and mixed-signal circuitry synthesis. The main contribution of this thesis is successful development of a method, which is able to handle complex multi-group symmetry, substrate sharing, and other topological constraints in the analog and mixed-signal layout placement design using *transitive closure graph* (TCG) representation.

This thesis proposes a set of symmetric-feasible conditions, which can guarantee symmetric placement of sensitive cells with respect to one or multiple symmetry axes for reduction of parasitic mismatch and thermal gradients. A new contour-based packing scheme has been developed with the time complexity of $O(p \cdot n \cdot lgn)$, where p is the number of the symmetry groups and n is the number of the placed cells. Furthermore, a set of perturbation operations is devised with the time complexity of $O(n)$, where n is the number of the placed cells, in order to generate a random symmetric-feasible TCG state from an existing one. The experimental results show the effectiveness and superiority of this proposed scheme compared to the other state-of-the-art placement algorithms for analog layout designs.

In addition, the proposed method is able to handle the substrate sharing constraints which require the devices to be placed adjacent to share a common substrate in order to decrease the effect of substrate coupling. To the best of the author's knowledge, this is the first proposed approach to handle the substrate sharing constraints based on topological representations. Other topological constraints such as

relationship, abutment and alignment can also be handled by our method.

The thesis first presents a brief introduction to the analog design, including electronic design automation, analog synthesis, and analog placement. The previous important works, which intended to solve the analog placement problem, are thoroughly surveyed and analyzed. Secondly the TCG-based method to handle complex analog layout constraints such as symmetry, substrate sharing and other requirements is detailed and the corresponding algorithm complexity is analyzed. Finally the performance of the proposed method is compared with the other alternatives and the conclusions are drawn.

Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor Dr. Lihong Zhang for his enthusiastic supervision during the past three years of research. I cannot finish this thesis without his personal guidance, intelligence, inspiration and encouragement while I pursued my interest in analog design. I also appreciate the financial support he provided as well as that from the School of Graduate Studies.

I am grateful to Dr. Ramachandran Venkatesan and Dr. Cheng Li for first providing me a chance to enter this University. I would also like to thank Dr. Theodore S. Novell for their teaching and guidance in my graduate studies.

Thanks to Tao Bian, Xueying Zhang, Liang Zhang, Lei Song, and Guan Wang for their generous support and friendship. I would like to thank everyone in this laboratory for their time, assistance and generous patience. Especially thanks to Pu Wang and Shi Chen who gave me a lot of advices and support either in academic or everyday life.

I am forever thankful to my parents and my wife, Yadan He. Thank you for your unending support, willingness to accept and endless love.

Finally, I would like to thank the Canadian Microelectronics Corporation (CMC) for providing the Cadence platform. Thanks to the support by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Contents

Abstract ..... **ii**

Acknowledgements ..... **iv**

List of Figures **4**

List of Tables **7**

Chapter 1 Introduction **8**

1.1 Analog Circuit Design and Electronic Design Automation 10

1.2 Analog Circuitry Synthesis and Layout Synthesis 14

1.3 Analog Placement 18

1.3.1 Analog Placement Problem *18*

1.3.2 Analog Placement Constraints *20*

1.4 Motivation, Contributions and Organization of the Thesis 22

Chapter 2 Prior Work **25**

2.1 Absolute Placement Method 26

2.2 Topological Methods 28

2.2.1 Sequence Pair (SP) *29*

2.2.2 Corner Block List (CBL) *35*

2.2.3 Tree Structures *38*

2.2.4 Graph Construct *41*

2.3 Summary 44

Chapter 3 Search Engine for the Analog Placement Problem **48**

3.1 Simulated Annealing Algorithm 48

3.2 Genetic Algorithm 52

3.3 Neutral Network 55
3.3.1 Introduction to Neutral Network 56
3.3.2 Analog Placement Problem Modeling 58
3.3.3 Algorithm Description 59
3.3.4 Neural Network Method Results 68
3.3.5 Neural Network Method Summary 72

3.4 Summary 72

Chapter 4 TCG-based Method to Handle the Symmetry Constraints 74

4.1 Symmetric-Feasible TCG 74

4.2 Symmetric Packing 80

4.3 Symmetric-Feasible TCG Perturbations 95
4.3.1 Cluster Edge Move-Reverse and Edge Move Operations 96
4.3.2. Five Perturbation Operations 99

4.4 Experimental Results 111

4.5 Summary 122

Chapter 5 Substrate-sharing and Other Constraints 124

5.1 Introduction 124

5.2 Substrate Sharing Problem Definition 130

5.3 Algorithm to Handle Substrate Sharing Constraints 132

5.4 Experimental Results of the Substrate Sharing Method 136

5.5 Other Constraints 139
5.5.1 Cell Relationship Constraints 140
5.5.2 Cell Proximity Constraints 142
5.5.3 Cell Alignment Constraints 143

5.6 Summary 144

Chapter 6 Conclusions and Future Work 145

6.1 Conclusions 145

6.2 Future Directions 146

References 150

Appendix - List of Publications 157

List of Figures

Figure 1.1 Analog design flow 12

Figure 1.2 Optimization and evaluation 15

Figure 1.3 The traditional layout design 17

Figure 1.4 Example of analog circuit and analog layout 19

Figure 1.5 Constraints generation and checking in the layout synthesis 21

Figure 2.1 (a) SP positive step-lines (b) SP negative step-lines 31

Figure 2.2 (a) horizontal constraint graph (b) vertical constraint graph 33

Figure 2.3 The symmetric feasible examples *(a, a')* and *(b, b')* are symmetric pairs: (a) $a' \vdash b,\ b' \vdash a'$; (b) $a' \vdash b,\ b' \vdash a'$;(c) $b \perp a,\ b' \perp a'$; (d) $b \perp a,\ b' \perp a'$; and (e) $a' \vdash b,\ b' \perp a'$ 34

Figure 2.4 CBL example 36

Figure 2.5 Example of O-tree 39

Figure 2.6 The HB*-tree representation and corresponding placement 40

Figure 2.7 (a) The placement and (b) the corresponding TCG 43

Figure 2.8 (a) The placement and (b) the corresponding TCG-S. 44

Figure 3.1 Overflow of the GA-based analog placement algorithm 55

Figure 3.2 Network Architecture 58

Figure 3.3 ANN placement algorithm 67

Figure 3.4 The result of a simple 4 cells placement problem 69

Figure 3.5 Final placement of Circuit1 using *MF* 71

Figure 4.1 (I) group Γ is placed at the left of group Φ; (II) Γ is placed below Φ; (III) Γ and Φ are intermingled; (IV) Φ is placed within Γ 77

Figure 4.2 (I) (b, b') are placed within (a, a'); (II) (a, a') are placed at the bottom of (b, b'); (III.a) $a \vdash b$, $b' \vdash a'$, $a \vdash c$, $c \vdash b$, $b' \vdash d$, and $d \vdash a'$; (III.b) $a \vdash b$, $a' \perp b'$, $a \vdash c$, $c \vdash b$, $d \perp b'$, and $a' \perp d$; 79

Figure 4.3 (a) TCG; (b) corresponding SP; (c) symmetric placement with separate self-symmetric halves; (d) final symmetric placement 82

Figure 4.4 Symmetric packing flow 85

Figure 4.5 Simplified Symmetric packing flow 90

Figure 4.6 Packing sub-sequences 92

Figure 4.7 Examples of the packing 94

Figure 4.8 Perturbation example 98

Figure 4.9 Symmetric cell move 103

Figure 4.10 Example of symmetric-cell move 104

Figure 4.11 Asymmetric cell move 105

Figure 4.12 Example of asymmetric move 108

Figure 4.13 Asymmetric cell move 109

Figure 4.14 Example of symmetric group move 110

Figure 4.15 Final placement of circuit ami33 120

Figure 4.16 Final placement of circuit lnamixbias_2p4g 121

Figure 4.17 The schematic of the OTA 121

Figure 4.18 Final placement of circuit OTA 122

Figure 5.1 An example of applying the substrate sharing constraint 125

Figure 5.2 Various forms of device geometry sharing (a)MOS diffusion merge (b) MOS well merge (c) MOS bulk contact merge (d) MOS gate abutment (e) MOS strapping abutment (f) BJT collector merging (g) BJT guard-ring merging (h) capacitor abutment. 128

Figure 5.3 Use of protection frames in merge detection (a) no merging (b) complete merging, no illegal overlap (c) merging and illegal overlap. 129
Figure 5.4 The substrate area of cells 131
Figure 5.5 An example of merging 131
Figure 5.6 Partial sharing and full sharing 132
Figure 5.7 Example of merge 133
Figure 5.8 Substrate sharing packing flow 134
Figure 5.9 Example of the packing flow 135
Figure 5.10 Placement of APTE 139
Figure 5.11 Placement of lnamixbias_ 2p4g 139
Figure 5.12 Example of relationship constraint 141
Figure 5.13 Example of ami33 with relationship constraints 141
Figure 5.14 Example of proximity constraint 142
Figure 5.15 Example of ami33 with proximity constraints 142
Figure 5.16 Alignment exmaple 144
Figure 5.17 Example of ami33 with alignment constraints 144

List of Tables

Table 2.1 Comparison of different topological representations in the context of symmetric-aware placement 47
Table 3.1 Neural network algorithm circuits results 71
Table 4.1 MCNC benchmarks 113
Table 4.2 Approaches for comparison 114
Table 4.3 Results of MCNC benchmarks 114
Table 4.4 Results of industry circuits 117
Table 4.5 Results of modified industry circuits and OTA 118
Table 4.6 SP with linear programming 120
Table 5.1 Results of MCNC benchmarks of substrate sharing constraints 137
Table 5.2 Results of Circuits of substrate sharing constraints 138

Chapter 1 Introduction

With the development of complex semiconductor technologies, very-large-scale integration (VLSI) circuits have constantly developed since 1970s. VLSI is the process to design and fabricate integrated circuits (IC) by combining a large number of transistor-based circuits into a single chip [1].

The first semiconductor chips only contained one transistor on each chip. However, the users required more individual functions, or systems, with more transistors to be integrated on the chips. Therefore, the integrated circuits became very demanding. The first integrated circuits consisted of a few diodes, transistors, resistors and capacitors to fabricate one or more logic gates on one single device. Then small-scale integration (SSI) and medium-scale integration (MSI) extended the IC devices with hundreds of logic gates. A process, known as large-scale integration (LSI), improved the size of systems to at least one thousand logic gates. Recently, the technology has developed significantly and today's microprocessors have many millions of gates and hundreds of millions of individual transistors to perform complex functions.

Nowadays, the VLSI circuits and systems have been widely used and they have achieved extremely high performance. As of early 2008, billion-transistor processors were commercially available. One typical example is Intel's Montecito Itanium chip. It shows that the semiconductor fabrication is moving from the current generation of 65

nm processes to the next 45 nm generation and facing new challenges such as increased variation across process corners. Its large transistor count is largely due to its 24 MB L3 cache. Another example is Nvidia's 280 series GPU, which is unique in the fact that 1.4 billion transistors are used for logic computation.

Due to the development of VLSI, current designs use extensive design automation and automated logic synthesis to lay out the transistors to achieve better performance and enable higher levels of complexity in the resulting logic functionality [2]. This modern design methodology is completely different from the earliest design. However, several certain logic blocks, like SRAM cells or analog building-blocks, are still designed by hand to ensure the highest performance and to avoid errors.

As a consequence of technology scaling and complexity of new microprocessors, VLSI design has encountered several challenges [3]. The first challenge is the power usage which is due to the fact that the threshold voltages have ceased to scale with advancing process technology. The second challenge is process variation. As lithography techniques approximate to the fundamental laws of optics, doping concentrations become extremely difficult and error-prone. Simulations across multiple fabrication process corners should be considered before the chip is certified ready for production. In addition, the timing and the first-pass success should be considered seriously due to the change of the physical features of the circuits. Abundance of stricter design constraints is another important aspect in the design. The constraints for layout become much more stringent and play a more important role due to lithography and etching issues with scaling. Many companies now opt to switch to electronic design

automation (EDA) [2] tools to automate their design process because of the overhead for custom design.

In this thesis, an efficient algorithm to handle complex constraints in the placement design is presented. Its features are analyzed theoretically and the performance is evaluated with promising experimental results. This chapter is to introduce the general concepts of the analog design, which provide the introduction and background of the whole thesis. Section 1.1 reviews the concept of the analog design and electronic design automation including the flow of general analog circuit design. Section 1.2 introduces two important steps in analog design, that is, analog circuitry synthesis and optimization. Section 1.3 details analog placement problem and lists the analog placement constraints. Section 1.4 raises the motivation and purpose of our analog placement research work, summarizes the main contribution of this work and presents the organization of this thesis.

1.1 Analog Circuit Design and Electronic Design Automation

Going from one single transistor to multimillion transistor circuits has provided the people with more functionality than the past generations of electronics. The microelectronics market, in particular, the markets for application-specific ICs (ASICs), application-specific standard parts (ASSPs), and high-volume commodity ICs are

characterized by an ever-increasing level of integration complexity, now featuring multimillion transistor ICs [3]. To handle the design complexity, hierarchical design and reuse of IP blocks tend to be inevitable. In recent years, complete systems that previously occupied one or more boards have been integrated on a few chips or even one single chip. Examples of such systems on a chip (SoC) are the single-chip TV, the single-chip camera [3], or new generations of integrated telecommunication systems that include analog, digital, and even radio-frequency (RF) sections on one chip.

Due to the rising level of integration, the complexity of integrated circuits increases. More and more functionality can be added as new processes of technology evolve [2]. With the increasing complexity, use of computer-aided design (CAD) tools that support design on a hierarchy of abstractions becomes more important. For digital circuit design, there exist a large variety of tools and design methodologies, which efficiently support designs using several levels of abstraction. This is required to keep in phase with the new capabilities offered by technology and design constraints. However, for analog circuit design, the situation is completely different. The level of abstraction is still kept at very low levels and there are not many efficient CAD tools available. The lack of a structured design flow is one of the major problems in analog circuit design. This problem becomes obvious when analog and digital circuits are combined on the same chip, as in the mixed-signal SoCs. In general, the digital parts account for about 90% of an integrated circuit while only 10% is analog. However, most of the time and efforts are spent on the analog design part [2].

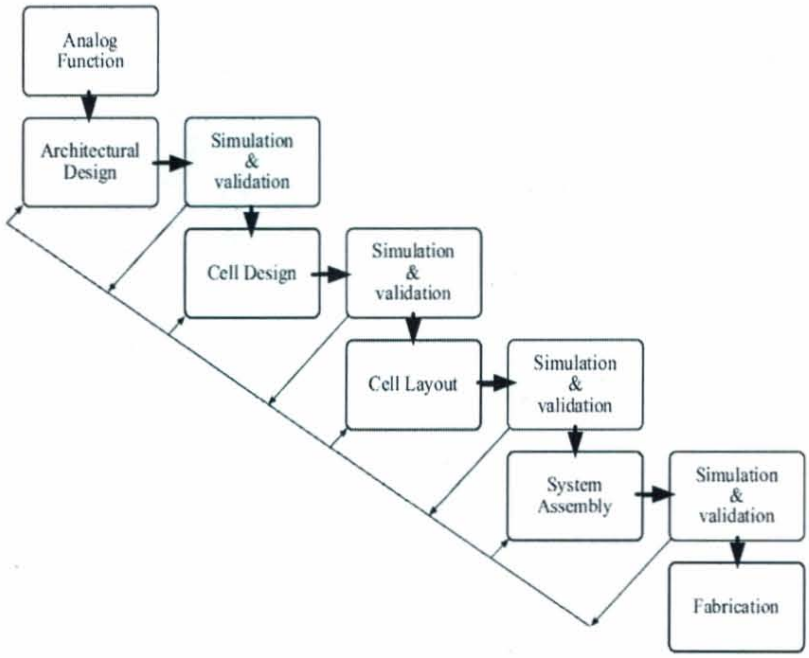


Figure 1.1 Analog design flow

A simplified view of an analog design flow is shown in Figure 1.1 [3]. The analog circuit designers normally start with an idea of the functionality to implement. The function is mapped onto an architectural description. In this process, the functionality is decomposed into a set of high-level building blocks. The decomposition is continued until the functional block can be mapped onto a set of lower-level analog building blocks. The simulations done at this level are typically carried out by using high-level models in order to validate the functionality of the concept.

From these simulations, the specification on the low-level blocks is extracted [3]. The performance specification contains requirements on the various performance

metric of the circuit. Here the performance is the measurement of properties, which is used to simulate the behaviour of a cell. In the next step, these cells are realized by designing the low-level building blocks that comply with the previously derived performance specification. The cell design step includes choosing between several possible realizations in order to implement the functionality in the most efficient way. During the layout phase, the geometry for the functional blocks is determined. Finally, the building blocks are assembled to implement the desired functionality. Throughout the design process, excessive simulations and validation steps are required. If the circuit fails to meet the specification at some level, the proceeding design steps must be revised as shown in the Figure 1.1. That may include backtracking several former steps in the analog circuit design process.

Electronic design automation (EDA) has become increasingly important in both digital and analog design. Typically the chip designers at semiconductor companies have to use a variety of EDA tools since large chips are too complex to design by hand. The EDA tools have rapidly increased in great importance with the continuous scaling of semiconductor technology. In addition, EDA software is used to evaluate an incoming design for manufacturing readiness as well as for programming design functionality into field-programmable gate arrays (FPGAs) [2].

Before the dawn of EDA, integrated circuits were designed by hand and manually laid out, thus required lots of time and resources. By the mid-70s, the designers had begun to use the automation with drafting. In the meantime, the first placement and routing tools were developed. The earliest EDA tools were designed academically, and

then used in industry in 1981. Many EDA companies such as Daisy Systems, Mentor Graphics and Valid Logic Systems were all founded around this time. The development of hardware description languages such as Verilog and VHDL permitted the simulators to directly report the simulation results of chip designs. This improvement also accelerated the development of EDA tools.

Current digital modular flows use the front ends to produce standardized design descriptions that compile into invocations of cells. Cells perform the logic or other electronic functions using a specific integrated circuit technology. Generally speaking, the fabricators provide libraries of components for their production processes, with simulation models that can fit standard simulation tools. However, analog EDA tools are much less modular since many more interference functions are required.

In the past decade, people have seen that the electronic design automation is starting to play a very important role in the analog design. However, on account of the complexity of analog circuit design and constraints, the EDA tools still need to be improved both in breath and in depth.

1.2 Analog Circuitry Synthesis and Layout Synthesis

Analog synthesis gets particularly significant for ever-growing mixed-signal SoC designs [4]. However, the analog design is an intrinsically difficult subject as it often

has to explore a much larger design space. As a matter of fact, in modern VLSI design the analog portion of a mixed-signal chip still has to be routinely handcrafted by experienced designers, which costs extraordinarily disproportional amount of efforts and time compared with only a small fraction occupied within the entire chip [5]. Figure 1.2 shows the optimization and evaluation flow of the analog circuit design. Here the optimization engine is to provide candidate circuit designs to the evaluation engine, whereas the evaluation engine is to evaluate circuit performance of the current candidate design in order to be prepared for the next run of optimization.

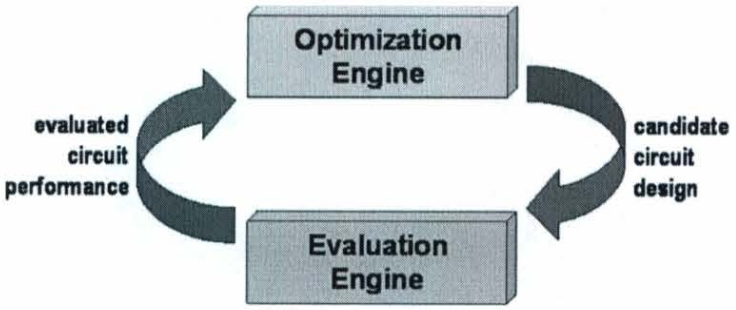


Figure 1.2 Optimization and evaluation

Recently, the commercial CAD tool has supported analog cell-level circuit and layout synthesis. Gielen and Rutenbar [2] offered a fairly complete survey of this area. The analog synthesis consists of two major steps: 1) circuit synthesis followed by 2) layout synthesis. Most of the basic techniques in both circuit and layout synthesis rely on powerful numerical optimization engines coupled to the evaluation engines that qualify the merit of the evolving analog circuits or layout candidates [5].

The layout synthesis relies more on combinatorial optimization techniques [6]. The layout of all devices will be determined so that the layout meets the given specifications with an optimal cost. Figure 1.3 shows the traditional flow of the layout design. Based on high-level constraints, three phases (i.e., placement, routing, and compaction) are included in the layout design. We can see adding the high-level constraints and generating the correct placement are the first steps for the analog layout generation.

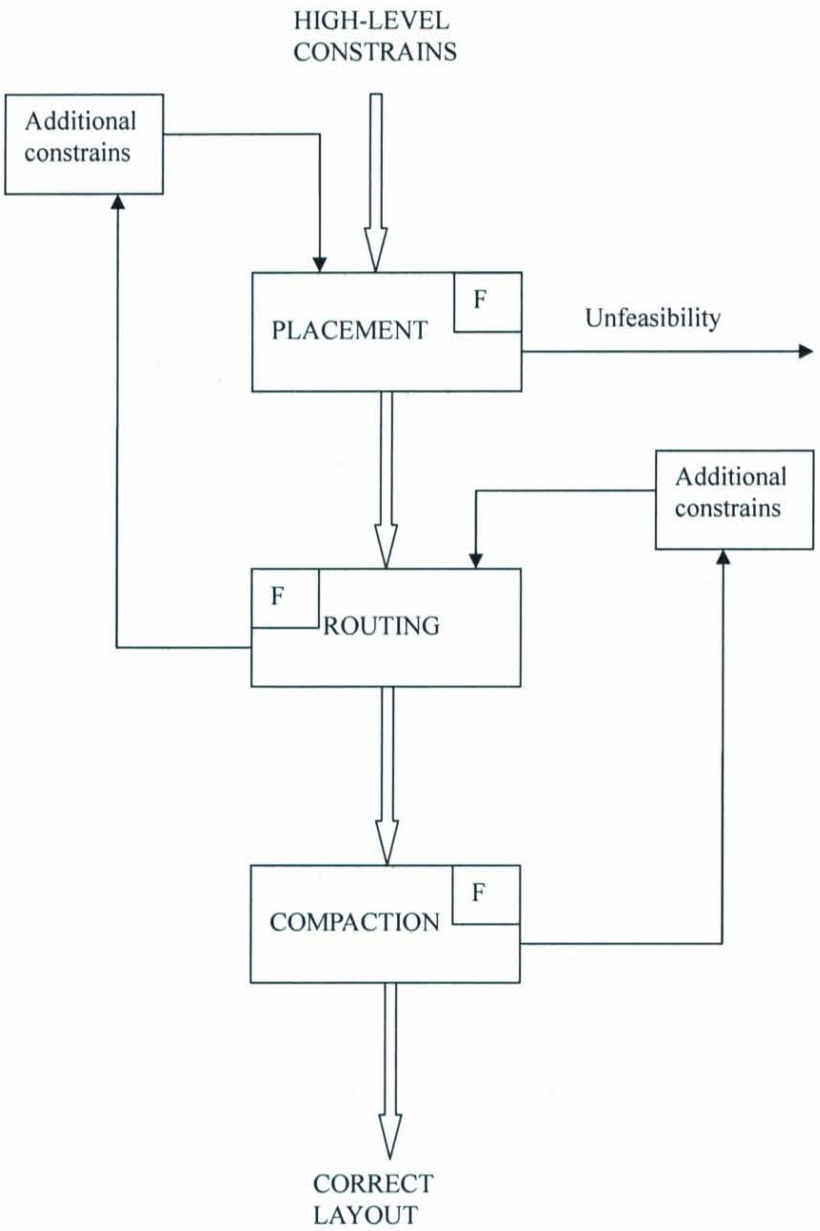


Figure 1.3 The conventional layout design

Although successful, these simulation-based analog synthesis methods still have to be used with care by designers because the processing time tends to be long and involves some errors against the constraints. To reduce the CPU time and the complexity, how to handle the complicated design constraints is an active area of research.

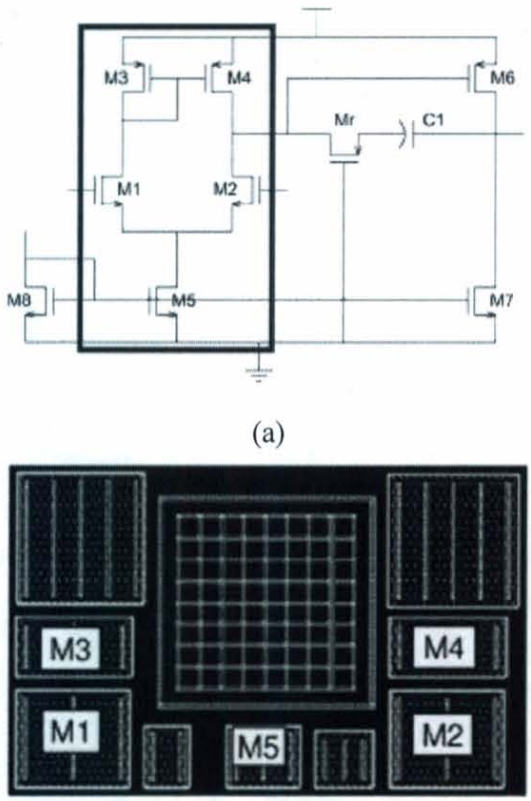
Several challenges have been presented for the synthesis and optimization in the research domain. Although the traditional methodology works well for circuits with the range of dozens of devices, for larger circuits the time required for each single simulation is too long. In addition, how to handle the analog layout constraints becomes increasingly important.

1.3 Analog Placement

1.3.1 Analog Placement Problem

Extensive research has shown that automatic analog layout generation is a nontrivial process in the analog synthesis [7]. In the analog layout design, the placement problem is about how to locate a set of rectangular or rectilinear cells on a plane. It has been well recognized as one of the most significant stages in the analog layout synthesis [8]. The goal of the placement design is that the placed cells do not overlap with one another and the total area is minimized with certain constraints (e.g.,

symmetry, substrate sharing, etc.) satisfied. Figure 1.4 shows an example of the analog placement. The schematic of the circuit is shown in Figure 1.4(a), and one type of layout style is shown in Figure 1.4 (b). It is not hard to see that the circuit elements are transferred to the rectangle blocks and thus the problem of placement is formulated [9].



(a)

(b)

Figure 1.4 Example of analog circuit and analog layout

Since placement is one of the most significant stages in the analog-layout synthesis [8], the study of the analog placement problem has attracted great interest in industry and academia [10]. As a matter of fact, once an analog placement solution is

fixed, most of the electrical effects are largely determined and more seriously, and some undesirable effects caused by a poor placement cannot be compensated by the following routing procedures. To meet the device-matching constraints in the design of analog layouts, it is usually preferable to cluster multiple devices to form parameterized cells or modules. Thus, the objective of the analog cell placement problem is to position cells appropriately so that the chip area and the total wire length of the interconnections can be minimized under the given constraints.

1.3.2 Analog Placement Constraints

In the analog placement design, certain constraints from the fabric analog circuits have to be imposed to improve the performance and meet specific requirements. As shown in Figure 1.5, constraint generation and checking are very important in the analog layout synthesis. In the following several important analog layout constraints, including the symmetry, substrate-sharing, relationship, abutment, and alignment constraints, will be introduced.

Symmetry, as an important type of topological constraints, widely appears in expert analog layouts. Symmetry constraints require sensitive cells to be placed on the opposite sides with respect to the corresponding symmetry axes in an identical or mirror manner. The symmetry constraints can reduce the effect of parasitic mismatch that may increase offset voltage and decrease power-supply rejection ratio. In addition,

they can reduce circuit sensitivity to thermal gradients and achieve better electrical properties. There are two types of symmetry constraints among cells: symmetric pairs and self-symmetric cells. A symmetric pair is a pair of cells placed in identical/mirror orientations and located on the opposite sides with respect to one particular symmetry axis. As for the second type of the symmetry constraints, self-symmetric cells need to be placed along the symmetry axis and share the same axis with the other symmetric cells. The cells, which include the symmetric pairs and self-symmetric cells sharing one common symmetry axis, are defined as one symmetry group. Multiple symmetry groups widely exist in the analog circuits [10].

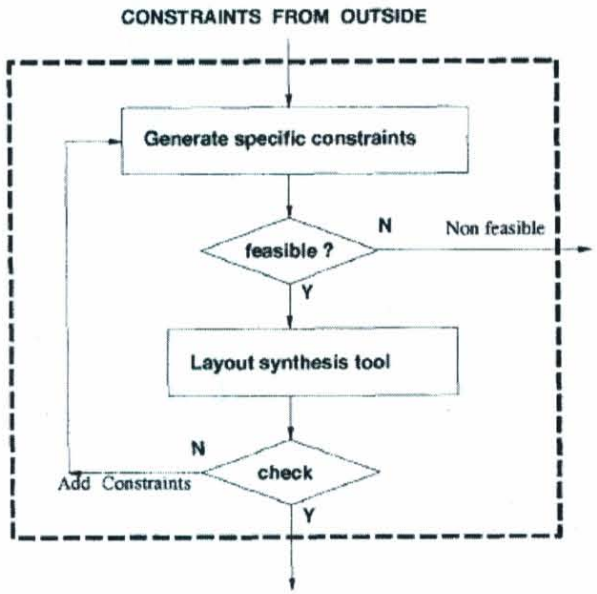


Figure 1.5 Constraints generation and checking in the layout synthesis

The substrate-sharing constraint limits devices to an adjacent placement so that

the devices can share a common substrate/well region. This can significantly decrease the effect of substrate coupling. For the substrate-sharing constraints, we can merge the substrates of the adjacent cells to minimize the cost of the area and the wire length of the placement. The performance of analog circuits can be improved by applying this type of constraints.

Moreover, there are some additional constraints, such as cell relationship, abutment, and alignment constraints, for the analog circuit design. In the practical analog layout design, the designers may have specific placement requirements that demand one or several cells to have particular topological relationship with another cell. This is called relationship constraint. Besides, the cell abutment constraints require some cells to be placed in an abutting manner. The cell alignment constraints require one cell to be located vertically or horizontally in alignment with other cells.

1.4 Motivation, Contributions and Organization of the Thesis

According to my survey, quite a few approaches [11]-[17] have been proposed to handle the analog placement problem. However, each method has its own features as well as drawbacks. Therefore, I am motivated to find a methodology which has better performance compared to previous work and is able to handle more complex analog placement constraints.

In this thesis, I propose a method using *transitive closure graph* (TCG) [11] to

handle the multiple symmetry-group constraints, substrate-sharing and other topological constraints. The proposed method is able to produce generic solutions to analog placements with efficient perturbation and packing schemes.

The major contributions of this thesis are listed as follows.

- Necessary and sufficient conditions, which can verify the symmetric feasibility of TCG representations in the context of general symmetric placement situations enclosing multiple symmetry groups, are introduced;

- I propose an efficient contour-based packing scheme to convert symmetric-feasible TCGs under multi-group symmetry constraints to symmetric placements in polynomial time;

- A set of random perturbation operations with the time complexity of $O(n)$ is defined to generate new multi-group symmetric-feasible TCGs;

- The proposed placement algorithm is able to cover all the possible topological situations. And the experimental results show that this method achieves superior performance compared to the other state-of-the-art work;

- I propose a method to handle the substrate-sharing constraints and apply the merging process to minimize the size of the placement. In addition, it can handle several other constraints such as cell relationship, abutment, and alignment.

The rest of the thesis is organized as follows. Chapter 2 conducts a brief review of previous works. These works are compared with each other and their instinct features are pointed out. In Chapter 3, I detail the search engine such as simulated annealing algorithm and genetic algorithm. I design an algorithm to solve the placement problem by using artificial neural network and compare it with the previous two algorithms. In Chapter 4, I introduce symmetric-feasible conditions using TCG to handle placement in the context of multiple symmetry groups. The contour-based packing algorithm, which is to construct a symmetric placement from a symmetric-feasible TCG, is detailed. And a set of perturbation schemes for maintaining symmetric-feasible TCGs are discussed. In Chapter 5, I define a solution to handle the placement problem with substrate-sharing constraints. The packing and perturbation methods are also proposed. The schemes to handle other constraints such as relationship, alignment and abutment are also discussed in this Chapter. Finally, I draw the conclusions and enumerate some future work in Chapter 6.

Chapter 2 Prior Work

In this chapter, some important prior works, which were targeted at the analog placement problem, will be first surveyed. Then they are evaluated to see whether they are suitable to handle the symmetry constraints. Finally a comparison of the previous methods is made and a direction for further development is proposed.

There are two streams for the placement methods: absolute placement and relative placement. For the absolute placement [10], cells are located by means of their absolute coordinates. This method has been traditionally regarded as the most effective solution to the analog placement in the past decades. The main drawback of the absolute placement lies in the fact that it may generate an infeasible placement with overlapping cells and thus requires a post-processing step to eliminate the overlap. In contrast, the relative placement (also known as topological placement) method is based on topological representations [9] [11]-[17]. Recently, it has drawn more attention from the researchers and a few topological representations, such as Sequence Pair (SP) [12], O-tree [13], B*-tree [14], Corner Block List [15], Transitive Closure Graph (TCG) [11], Transitive Closure Graph with topological order (TCG-S) [16], and hierarchical B-tree (HB*-tree) [9] [17], have been applied to solve analog

placement problem.

2.1 Absolute Placement Method

The absolute placement method locates the cells based on their coordinates. Its main flow based on a stochastic scheme (normally simulated annealing algorithm (SA) [18]) firstly employs a constraint generator to analyze the constraints. A cost function like (2.1) is used to evaluate the merit of one placement state.

$$Cost = \sum_i \beta_i \cdot Parameter_i \qquad (2.1)$$

where β is the weight of the parameters to the cost. The parameters include the different aspects of the cost value, such as wire length, area cost, constraint plenty, distance of cells, and so on.

The absolute representation was first introduced by Jepsen and Gellat [19], where the cells are specified in terms of *absolute* coordinates on a gridless plane. The moves are simple coordinate shifts or changes in cell orientation. Cells are allowed to overlap in possibly illegal ways, as no restriction is made to refer to the relative position of a cell with respect to another cell. A (weighted) penalty cost term is associated with infeasible overlaps, and this penalty must be driven to zero in the optimization process. The absolute representation is well suited to handle device matching and symmetry constraints, typical to analog layouts. It is also allowed to

explore the beneficial device overlaps. Taking these factors into consideration, the absolute representation was the choice for KOAN/ANAGRAM II [20], PUPPY-A [10], and LAYLA [21] systems. However, this representation has revealed a drawback. Due to the complexity of the cost function, the total (infeasible) overlap in the final placement solution is not necessarily equal to zero: a final step eliminating gaps and overlaps must be performed, degrading the computation time and the solution optimality (in terms of the cost function). Moreover, the weight of the overlap term must be carefully chosen [10]: if it is too small, the cells may have the tendency to collapse; if it is too large, the search ability of the optimization engine for a good placement (in terms of area, total net length, etc.) may be impeded. To combat this effect, an earlier version of the Timber Wolf system [22] used a sophisticated negative control scheme to determine the optimum values of the cost term weights. The absolute representation approach trades off a larger number of annealing moves to build layout configurations more quickly, which may not be always physically realizable.

Reference [10] provides an algorithm of the absolute placement method to handle the symmetry constraints. It presented a quantitative approach to automatically generate symmetry constraints. Symmetry is recognized as a particular case of matching between devices or interconnections belonging to distinct differential signal paths, which become effective when the circuit is operated in differential mode. A graph-based search algorithm, described in detail in [10], has been designed for the automated detection of all symmetry constraints. First, a graph is built, with a node

for each circuit net, and an edge for each device, to represent the circuit connectivity. Then all virtual grounds are detected by comparing the common and differential-mode gains of all nets. The search algorithm recognizes all the sub-graphs whose structure has the following characteristics:

1) Symmetric topology;

2) Matching constraints between symmetric graph elements;

3) The two halves of the structure are connected with one another by one or more real or virtual ground nets.

Each of these sub-graphs is a differential structure, and the symmetry constraints are all recognized as the matching constraints.

This method worked efficiently as a placement solution in the old times. However, it may generate the infeasible solution to the placement problem. Therefore, the intrinsically high complexity and the requirement of post-process step have made the researchers be recently focused on new topological methods.

2.2 Topological Methods

The second class of placement representations—named *topological*—allows trading off more complex, physically correct, and layout constructions per each random move for a smaller number of moves. Generally, the placement algorithms develop a packing scheme to transfer the representations to placements, and design a

set of perturbations to randomly generate new placements from the existing ones. Then the search engines, such as SA, genetic algorithm (GA), are used to obtain the optimized placement based on the predefined cost functions. In the following parts, several different types of the topological representations are introduced.

2.2.1 Sequence Pair (SP)

Sequence-pair [12] is considered to be a general representation for the placement problem. It uses an ordered pair of sequences (i.e., α- and β- sequences) to encode a placement. The topological relationship between two cells a and b can be derived from the SP representation as follows [12]:

$$\text{if } \alpha_a^{-1} < \alpha_b^{-1} \text{ and } \beta_a^{-1} < \beta_b^{-1}, \text{ then cell } a \text{ is to the left of cell } b; \quad (2.2)$$

$$\text{if } \alpha_a^{-1} < \alpha_b^{-1} \text{ and } \beta_b^{-1} < \beta_a^{-1}, \text{ then cell } b \text{ is on the below of cell } a, \quad (2.3)$$

where α_a^{-1} and α_b^{-1} represent the order of cells a and b in the α sequence, while β_a^{-1} and β_b^{-1} represent the order of cells a and b in the β sequence, respectively.

In [12], to generate the α-sequence, it needs to process SP positive step-lines as shown in Figure 2.1(a). For each cell, we draw lines using a pebble (which is regarded as the nib of a pen). The pebble is initially located at the upper right corner of cell x and starts to move upward. It turns its direction alternatively right and up until it

reaches the upper right corner without crossing: 1) boundaries of other cells, 2) previously drawn lines, and 3) the boundary of the chip. The drawn line is called the *up-right step-line* of cell x. Similarly, the *down-left step-line* of x is drawn. The union of these two step-lines and the connecting diagonal line of cell x is called the *positive step-line* of x. It is always possible to draw such positive step-line for a cell. They are referred to by the corresponding cell names. Follow this procedure, we can draw the α-sequence of the placement, which is (*e, c, a, d, f, b*).

Simultaneously, we have to obtain the β-sequence following a negative step-line process as shown in Figure 2.1 (b). The difference is that a negative step-line is the union of the *left-up step-line* and *right-down step-line*, whose direction changing policies are "left, up, left, up,...," and "right, down, right, down, ...," respectively. We order the negative step-lines also from left. Let the β-sequence be the cell name sequence in this order. So after the negative process, the β-sequence of the placement shown in Figure 2.1 (b) is (*f, c, b, e, a, d*).

To transfer a sequence-pair to a placement, reference [12] details a packing scheme for the sequence-pair. It needs to construct two constraint graphs of the packing scheme as shown in Figure 2.2 and the example is the same as Figure 2.1. The constraint of the sequence pair is detailed in the formulae (2.1) and (2.2). It is easily observed that the constraint imposed on the packing by a sequence-pair is unique and the constraints are always satisfied.

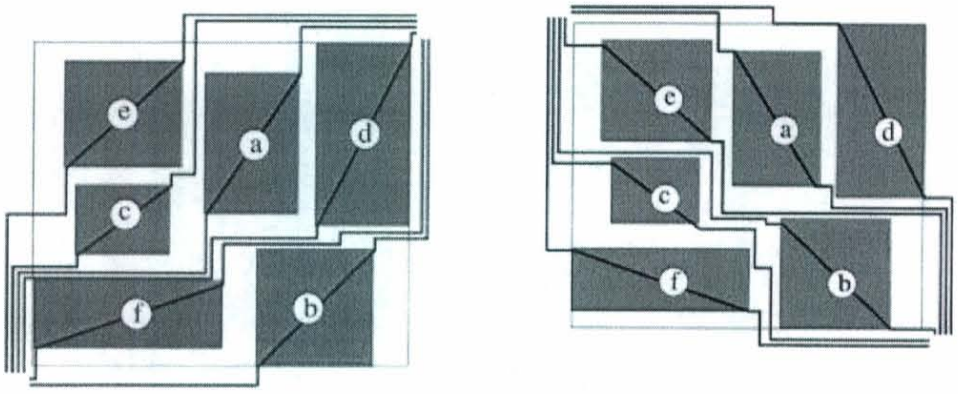
Given a sequence pair, one of the optimal packing under the constraints can be obtained in time $O(n^2)$ where n is the number of cells. By applying the well-known

longest path algorithm to directed acyclic graphs with weighted vertices, we can process the packing of the SP scheme. The process is given below. Based on "left of" constraint of the sequence pair, a directed and vertex-weighted graph $G_h(V, E)$ where V presents the vertex set, and E presents the edge set, called the *horizontal-constraint graph*, is constructed as follows.

1) V: source s, sink t, and vertices n labeled with cell names.

2) E: (s, x) and (x, t) for each cell x, and (x, x') if and only if x is on the left of x'.

3) Vertex-weight: zero for s and t, width of cell for the other vertices.



(a) (b)

Figure 2.1 (a) SP positive step-lines (b) SP negative step-lines

Similarly, the *vertical-constraint graph* is constructed using "below" constraints and the height of each cell. Neither of these graphs contains any directed cycle. We set the X-coordinate to be the longest-path length from s to x in G_h. The Y-coordinate of x is set independently using G_v. As shown in Figure 2.2, if two cells x and x' are in

horizontal relation, then there is an edge between x and x' in G_h; hence they do not overlap horizontally in the resultant placement. Similarly, if x and x' are in vertical relation, they do not overlap vertically. Thus no two modules overlap with each other in the resultant placement because any pair of cells is either in horizontal or vertical relation. The width and the height of the chip are determined by the longest-path length between the source and the sink in G_h and G_v, respectively. Since the width and the height of the chip are independent minimum, the resultant packing is the best of all the packing solution under the constraints. The longest-path length calculation in each graph can be done in $O(n^2)$ time, proportional to the number of edges in the graph. Following this process, we can obtain the final placement of the sequence pair. And the SP perturbation is very straightforward, just to change the order of the cells in the sequence with time $O(1)$.

To facilitate the description within this thesis, we will first present one definition in the following.

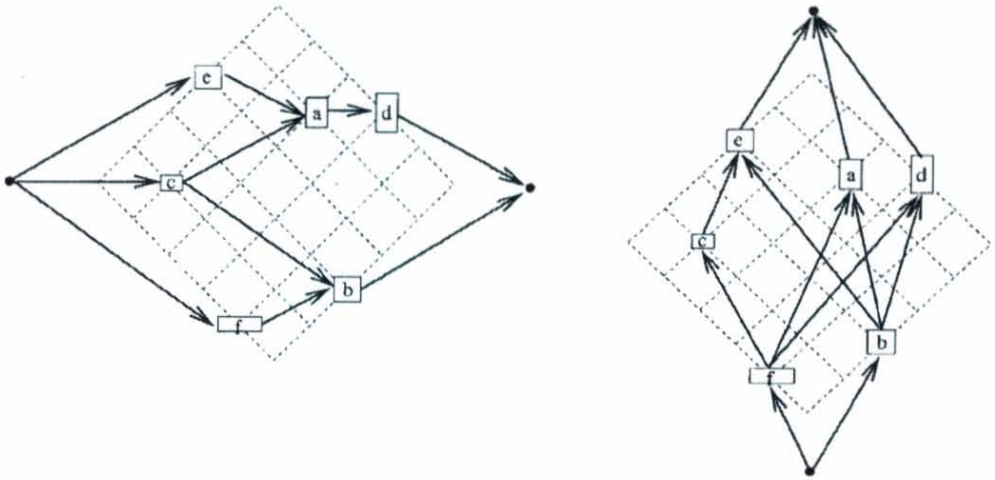
Definition 1: If a set of representations following a certain structure is called *symmetric-feasible*, then any symmetric placement can be expressed by a representation in that set while a representation in that set can guarantee to construct a valid symmetric placement.

References [23] and [24] developed two schemes to handle symmetric placement based on a set of the SP symmetric-feasibility conditions (advocated in [23]) as

follows:

$$\alpha_a^{-1} < \alpha_b^{-1} \text{ and } \beta_b^{-1} < \beta_{a'}^{-1}, \qquad (a, a') \in \Gamma, (b, b') \in \Gamma, \tag{2.4}$$

where Γ is one symmetry group.



(a) (b)

Figure 2.2 (a) horizontal constraint graph (b) vertical constraint graph

However, these symmetric-feasibility conditions have involved some intrinsic problems which are indicated in [25]. According to the conditions above, the symmetric cells should appear in a mirror form in two sequences of SP. The α- and β-sequences are indeed ordered in a mirror form. Thus, the SP representation above is symmetric feasible. However, this observation is partial, as revealed in [28]. For

instance, the SP of a symmetric placement depicted in Figure 2.3 (b) is *(a, c, b, b', d, a') (a, c, b, b', d, a')*, which is in line with the proposed formula (2.3). *(a, a')* and *(b, b')* are two symmetric pairs. Although another placement depicted in Figure 2.3 (d) is symmetric for *(a, a')* and *(b, b')*, its SP, which is *(a, c, b, b', d, a') (b, c, a, b', d, a')*, obviously does not satisfy formula (2.3). Therefore, searching in a subset of the symmetric-feasibility space as [23] may lead to a non-optimal solution.

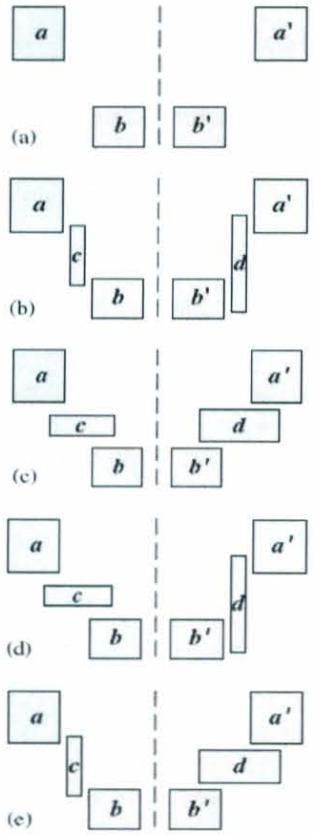


Figure 2.3 The symmetric feasible examples *(a, a')* and *(b, b')* are symmetric pairs: (a) $a' \vdash b, b' \vdash a'$; (b) $a' \vdash b, b' \vdash a'$;(c) $b \perp a, b' \perp a'$; (d) $b \perp a, b' \perp a'$; and (e) $a' \vdash b, b' \perp a'$

By enumerating a number of intrinsic problems in those SP symmetric-feasibility conditions (2.3), Kouda *et al.* proposed one linear-programming based scheme using SP [26] [27]. Although the formulation is general, the method has high packing complexity due to the nature of linear programming. Nevertheless, the work fits well into the framework of template-driven analog layout retargeting and optimization, which is an increasingly important area of research [29].

2.2.2 Corner Block List (CBL)

Corner Block List (CBL) [15] is composed of three sequences (*S, L, T*), where sequence *S* represents cell names from left-bottom corner to right-top corner, list *L* records the orientation of cell deletion operations, and *T* is the number of T-junctions uncovered if the corner cell is deleted from the packing. This representation features only linear time for constructing a placement if given a CBL.

In the CBL method [15], it firstly determines the orientation of corner block cell, and then processes the corner block deletion. If the comer block is horizontally oriented, to delete the comer block, we shift its left segment to the right boundary of the chip, and pull the attached T-junctions along with the segment. A T-junction is composed of two segments: a non-crossing segment and a crossing segment. If the comer block is vertically oriented, we shift its bottom segment to the top boundary of the chip, and pull the attached T-junctions along with the segment. The comer block

list is constructed from the record of a recursive comer block deletion. For each block deletion, we keep a record of block name, comer block orientation, and number of T-junctions uncovered. At the end of deletion iterations, we concatenate the data of these three items in a reversed order. Thus, we have the sequence S, orientations list L, and T-junction information list T. Each string of 1s is ended with a 0 to separate from the record of the next comer block deletion. We take the floorplan of Figure 2.4 as an example. First, cell d is deleted since d is vertical oriented and there is one T- junction attached at the bottom edge of block d. Thus, we keep a record (d, 0, 10). Cells a, b, g, e, c are deleted successively. We concatenate these records in a reverse order of deletion and derive a comer block list (S, L, T), where S = ($fcegbad$), L = (001100), and T= (001010010).

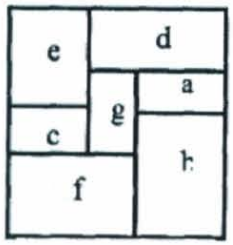


Figure 2.4 CBL example

The proposed CBL placement method of [15] is based on simulated annealing algorithm. The perturbations of the CBL are listed as follows.

1) Randomly exchange the order of the blocks;

2) Randomly choose a position in L, change *1* to *0*, or *0* to *1*;

3) Randomly choose a position in T, change *1* to *0* or *0* to *1*;

4) Rotate the cell;

5) Reflect the modules in both horizontal and vertical orientations;

6) Randomly choose a cell, then randomly choose an alternate cell to substitute the original one.

Using the algorithm above, among the random comer block lists one optimal solution can be generated. The algorithm works efficiently since the time complexity of transforming a comer block list to a placement configuration is $O(n)$. The number of the combinations of comer block list is $O(n!2^{3n-3}/n^{1.5})$. In addition, corner block list takes $n(3+\lceil lgn \rceil)$ bits to describe, where $\lceil lgn \rceil$ denotes the minimum integral number which is not less than lgn. Furthermore, comer block list represents the floorplan independent of the cell sizes. One can use this representation to optimize the cells with multiple configurations of widths and heights.

Liu *et al.* investigated the application of the CBL representation in the symmetry-aware context [30]. To handle the symmetry constraints, one need to build up a tree structure, which all of the later processing operations are based on. However, no detailed complexity analysis and experimental comparison with other prior work is provided in [30]. And that work only focuses on the situation of single symmetry-group constraints. Moreover, similar to the other tree representations, the CBL representation fails to effectively expose the topological relationship between any two cells before packing.

2.2.3 Tree Structures

Some tree structures, such as O-tree [13] and B*-tree [14], are employed to reduce high packing complexity for the analog placement problem. O-tree uses the ordered tree to represent the placement as shown in Figure 2.5 and traverse the tree using depth-first-search (DFS) order [13]. However, this representation can only handle compact placements rather than any general placements. In contrast, B*-tree is based on ordered binary trees to derive the admissible placements [14]. For a node, its left and right child nodes represent a left-right and bottom-top geometric relationships, respectively. However, for those tree representations, due to lack of the definition of the geometric relationship between each pair of cells, there is no awareness of cell relative positions and optimality before packing especially under complex constraints, resulting in degradation of solution quality and even longer execution time.

Most recently, HB*-tree [9] has been proposed for solving symmetry constraints using a concept of *symmetry island*. The author points out that when the symmetric cells of the same symmetry group are placed tightly (or even adjacent) to each other, the layout style is generally considered much better. Consequently, the sensitivities due to process variations can be minimized, and the circuit performance can be improved. So [9] defined the symmetry island, a placement pattern with respect to a symmetry group in which each cell in the group abuts at least one of the other cells in

the same group, and all the cells in the symmetry group form a connected placement.

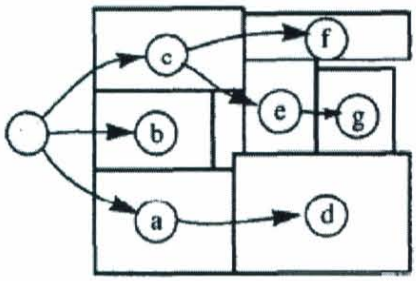


Figure 2.5 Example of O-tree

It uses a binary tree to represent a compact placement just like B*-tree. The root of a B*-tree corresponds to the cell (or called module) on the bottom-left corner. For each node *n* corresponding to cell *b*, the left child of *n* represents the lowest adjacent cell on the right side of *b*, while the right child of *n* represents the first cell above *b* with the same horizontal coordinates. The symmetric cells in one symmetry group are formed as one symmetry island to be represented by one hierarchy node. For example, in Figure 2.6 (a), there are two symmetry groups. The HB*-tree representation of this placement is shown in Figure 2.6 (b). The asymmetric cells placed adjacent to cells in the symmetry island on the top are connected as the right child to the contour nodes of the hierarchy node.

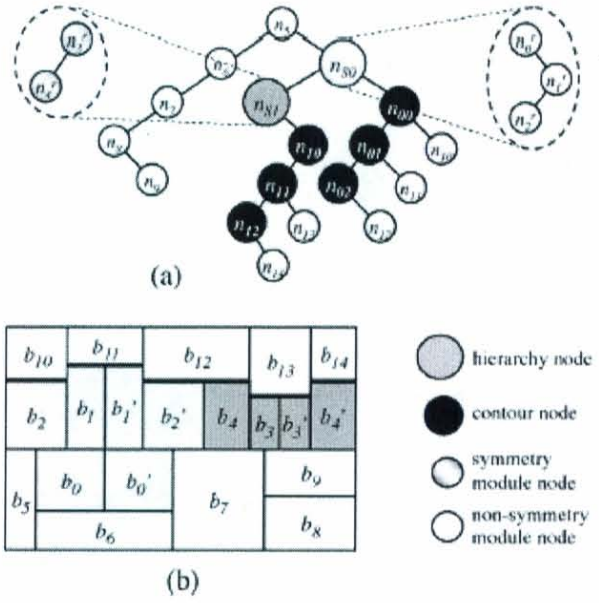


Figure 2.6 The HB*-tree representation and corresponding placement

The packing scheme of HB*-tree has two steps, that is, island packing and the HB*-tree packing. The packing of the island is similar to that of the B*-tree [14], which follows the *preorder-tree-traversal* procedure to calculate the coordinates of the cells. During the packing flow, two double-linked lists are implemented to keep both horizontal and vertical contour structures. After obtaining the coordinates of all representative cells in the symmetry group, we can calculate the coordinates of the symmetric cells and the extended contours. The HB*-tree packing also adopts the preorder-tree-traversal procedure. When a hierarchy node is traversed, the island in the hierarchy node should be packed first to obtain the contours of the symmetry island described previously. The contours are then stored in the corresponding

hierarchy node. When packing a hierarchy node that represents a symmetry island, we should calculate the best packing coordinate for the bottom boundary of the symmetry island. We then proceed to pack the left child of the hierarchy node. After the left child and all its descendants are packed, we pack the first contour node of the symmetry island, followed by the second one, and so on. When packing the contour nodes, we only need to update their coordinates and replace the hierarchy node in the contour data structure. The perturbation of the HB*-tree is based on the node operations and takes *O(lgn)* time due to the feature of tree structure.

The work of [9] is the first approach, which can handle symmetric packing with linear time complexity. However, it can only handle the situation where symmetric cells belonging to the same symmetry group are closely adjacent to each other (i.e., no asymmetric cell keeping symmetric ones separate). In addition, as it cannot denote the relationship between any two nodes before traversing the tree and packing, it is hard to handle any relative topology constraints (explicitly imposed by the designers), which require one cell (or symmetry group) to be placed on the left/right or bottom/top of another cell (or symmetry group).

2.2.4 Graph Structure

Graph structures have also been used to solve the analog placement problem. Assume there are a set of vertices V and a set of edges E, and a directed acyclic graph

$G = (V, E)$ is formed. *Transitive Closure Graph* (TCG) [11] of G is defined as graph $G' = (V, E')$ with $E' = (v_i, v_j)$. If $\forall (v_i, v_j) \in V$, there is a path from vertex v_i to v_j in G. A TCG representation of a placement consists of two graphs: horizontal transitive closure graph G_h and vertical transitive closure graph G_v. In both graphs, a vertex (e.g., v_i) corresponds to a cell. A direct edge $<v_i, v_j>$ in the G_h (G_v) means that cell v_i should be placed at the left (bottom) of cell v_j. This relationship is presented as $v_i \vdash (\perp) v_j$. For an edge $<v_i, v_j>$ in G_h (G_v), v_i (c_i) is called a *fan-in vertex* (*fan-in cell*) of v_j (c_j), whereas v_j (c_j) is called a *fan-out vertex* (*fan-out cell*) of v_i (c_i). The weight of vertex indicates width (height) of the corresponding cell. And the number of the *fan-in (fan-out) vertexes* of a cell is called in-degree (out-degree) of this cell. As shown in Figure 2.7 (a), the compact placement, which cannot be presented by O-tree, can be correctly presented by the TCG graphs.

The TCG placement scheme proposed in [11] needs to be implemented with some extra operations (e.g., checking conflict edges upon random perturbation) to guarantee the transitive closure feature (that is, if $v_1 \vdash (\perp) v_2$ and $v_2 \vdash (\perp) v_3$ holds, there must exist $v_1 \vdash (\perp) v_3$ in the graphs during perturbation). Thus, the complexity of the proposed perturbation and packing operations tends to be high. For the TCG packing scheme, the original TCG method uses the longest path algorithm for the graphs, which has the time complexity of $O(n^2)$.

In contrast, to handle the symmetry constraints, [31] deployed another graph-based topological representation called TCG-S [16] as shown in Figure 2.8, which combines TCG with a topological sequence. This TCG-S method developed a

contour-based packing method with the time complexity of *O(nlgn)*. The topological sequence is the same as the β-sequence in the SP method. It tunes the coordinates of the symmetric cells during the packing flow based on the symmetric constraints. Similarly, mandatory validity checking of TCG reduction edges significantly increases the perturbation complexity. Moreover, this proposed TCG-S method is only limited to handling the situation with one single symmetry group.

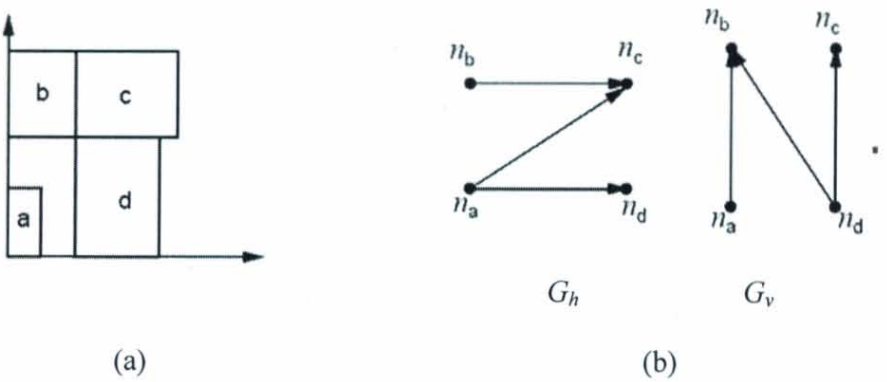


Figure 2.7 (a) The placement and (b) the corresponding TCG

To cover any possible symmetry situations and obtain a general solution to analog constraints, a preliminary work of [28] derived basic symmetric-feasible TCG conditions. But that work only exhibits the processing details for one single symmetry group and experiences high packing time complexity of $O(n^2)$, where n is the number of the placed cells. In this thesis, I am motivated to explore the TCG placement strategy to handle the complex constraints with multiple symmetry groups but with less packing and perturbation complexity.

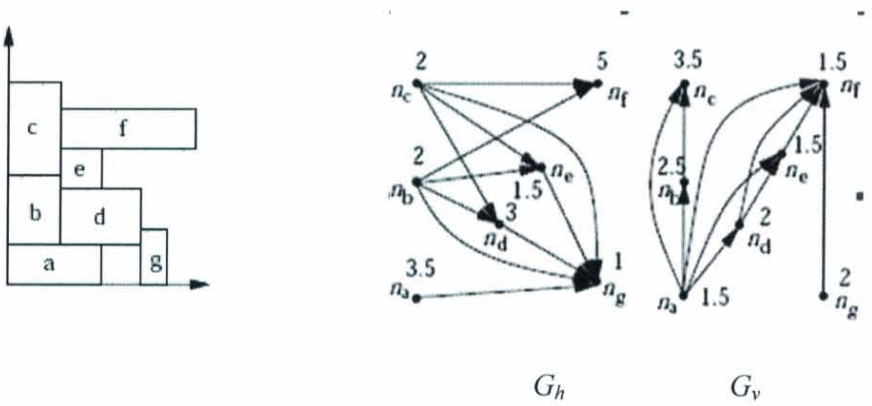


Figure 2.8 (a) The placement and (b) the corresponding TCG-S.

2.3 Summary

In this chapter different approaches for analog layout placement problem have been reviewed. Based on the comparison of the distinct methods above, the best way of guaranteeing the validity and efficiency of solution searching tends to be the topological methods. Considering the complex analog placement constraints as well as the features of TCG, I choose the TCG as the basic representation scheme in this thesis.

As two outstanding topological representations, TCG and SP are exhibited equivalent from the perspective of functionality [28]. As a matter of fact, they can be

converted to each other if either is available. Nevertheless, TCG and SP are different in the following aspects. Since SP is a sequence based representation, it is easy to be implemented and quite straightforward for the perturbation operations. One can simply perturb the sequence and obtain a new valid SP. In contrast, simple edge move (i.e., moving an edge from G_h to G_v or vice versa) or edge reverse (i.e., reversing an edge in G_h or G_v) operation for TCG is not always safe as the transitive closure feature of TCG cannot be automatically preserved. In this thesis, we will prove that TCG, if following our proposed perturbation scheme, is able to manage the perturbation while preserving the transitive closure with linear time complexity.

On the other hand, SP can only show the abutting geometric relationship between cells, while TCG is a graph-based representation that can use edge weights to store the distance information of cells. This feature makes TCG effortless to indicate the geometric relationship of the cells and further handle some additional constraints. Provided that the weight of an edge between two vertices in a TCG represents the width or height of the corresponding cell, cell separation (i.e., two cells are kept away from each other with a certain distance) and cell merging (i.e., two cells overlap with each other to share some regions such as bulk contact rows) constraints can be represented by having the edge weight more than or less than a pre-set value. Therefore, despite a topological representation, TCG is capable of obtaining a denser layout solution (i.e. due to cell merging) without compromising parasitic (i.e. separation of certain cells that include sensitive nets). In contrast, it is very hard to perform cell separation or cell merging with the SP representation.

In summary, the SP and TCG can be transferred to each other or complete the same perturbation operation in $O(n)$ time, where n is the number of the cells in the placement. Our study shows both representations mainly differ in the edge weight, which we can take advantage of in the analog placement design. The TCG-based symmetry-aware placement algorithm proposed in this thesis can be readily rephrased in terms of the SP representation. For the sake of any further development targeting at the analog placement problem, we propose the method based on TCG to handle the multiple symmetry-group constraints, the substrate sharing constraints, and some other constraints.

In addition, the features of different approaches, including SP [23], SP with linear programming (LP) [26], SP with dummy node (DN) [24], HB*-tree [9], CBL [30], TCG-S [11] and S-TCG (this work), are summarized in Table 2.1. The third column *Non-compact placement* shows whether the method is able to obtain the solution that is not compact. And the fifth column *Completeness* means whether a complete solution space is able to be explored by the individual approach. Our work in this thesis is denoted in the last line.

Table 2.1 Comparison of different topological representations in the context of symmetric-aware placement

Approach		Packing	Perturbation	Non-compact	Multi-sym	Completeness
SP	*general*[12]	$O(n^2)$	*O(1)*	No	-	yes
	Sym-SP[13]	$O(n^2)$	*O(1)*	No	yes	no
	SP with LP[26]	$O(n^2)$	*O(1)*	No	yes	no
	SP with DN[24]	$O(n^2)$	*O(1)*	No	yes	no
HB*-tree	*General* [14]	*O(n)*	*O(lgn)*	Yes	-	yes
	Symmetric[9]	*O(n)*	*O(lgn)*	Yes	yes	no
CBL	*general*[15]	*O(n)*	*O(n)*	No	-	yes
	symmetric[30]	-	-	No	no	no
TCG-S	*general*[16]	*O(nlgn)*	$O(n^2)$	Yes	-	yes
	symmetric[31]	$O(n^2)$	$O(n^2)$	Yes	no	no
TCG	*general*[11]	$O(n^2)$	$O(n^2)$	Yes	-	yes
	symmetric(this work)	$O(p \cdot nlgn)$	*O(n)*	Yes	yes	yes

Chapter 3 Search Engine for the Analog Placement Problem

Simulated annealing algorithm (SA) and genetic algorithm (GA) are two effective search engines for the placement approaches [32], because they can readily regard distinct constraints in different applications. SA and GA have been widely used for the layout synthesis of both digital [33], [34] and analog circuits. They typically can yield good placement results.

As an important part of our algorithm, we introduce the SA method in Section 3.1. The other important approach GA is detailed in Section 3.2. As one of my prior work, in Section 3.3 I propose an analog placement scheme based on the artificial neutral network, which is compared with the SA and GA algorithms. In the last section I make a brief summary.

3.1 Simulated Annealing Algorithm

Simulated annealing (SA) [18] is a generic probabilistic heuristic for the global optimization problems of applied mathematics, namely locating a good approximation to the global minimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For

certain problems, simulated annealing may be more effective than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution within a fixed amount of time, rather than the best possible solution.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one [18].

By analogy with this physical process, each step of the SA algorithm [18] replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called temperature), which is gradually decreased during the process. The dependency is that the current solution almost randomly changes when T is large, but increasingly "downhill" as T goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local minima — which is the bane of greedier methods.

In the earliest days of scientific computing, SA was introduced and could be used to provide an efficient simulation of a collection of atoms in equilibrium at a given temperature. In each step of this algorithm, an atom is given a small random displacement and the resulting change, $\triangle E$, in terms of the energy of the system is computed. If $\triangle E \leq 0$, the displacement is accepted, and the configuration with the

displaced atom is used as the starting point of the next step. The case $\triangle E>0$ is treated probabilistically: the probability that the configuration is accepted is $P(\triangle E) = exp(-\triangle E/K_BT)$. Random numbers uniformly distributed in the interval $(0, 1)$ are a convenient means of implementing the random part of the algorithm. One such number is selected and compared with $P(\triangle E)$. If it is less than $P(\triangle E)$, the new configuration is retained; if not, the original configuration is used to start the next step. By repeating the basic step many times, one simulates the thermal motion of atoms in thermal contact with a heat bath at temperature T [18]. This choice of $P(\triangle E)$ has the consequence that the system evolves into a Boltzmann distribution.

Using the cost function in place of the energy and defining configuration by a set of parameters $\{x_i\}$, it is straightforward with the Metropolis procedure to generate a population of configurations of a given optimization problem at some effective temperature [18]. This temperature is simply a control parameter in the same unit as the cost function. The simulated annealing process consists of first "melting" the system being optimized at a high effective temperature, then lowering the temperature by slow stages until the system "freezes" and no further changes occur. At each temperature, the simulation must proceed long enough for the system to reach a steady state. The sequence of temperatures and the number of rearrangements of the $\{x_i\}$ attempted to reach equilibrium at each temperature can be considered an annealing schedule.

Annealing, as implemented by Metropolis procedure, differs from integrative

improvement in that the procedure does not get stuck since transitions out of a local optimum are always possible at nonzero temperature [18]. A second and more important feature is that a sort of adaptive divide-and-conquer occurs. Gross features of the eventual state of the system appear at higher temperatures; fine details develop at lower temperatures.

Statistical mechanics contains much useful information for extracting properties of a macroscopic system from microscopic averages. Ensemble averages can be obtained from a single generating function, the partition function, Z,

$$Z = T_r \exp\left(\frac{-E}{K_B T}\right) \tag{3.1}$$

In which the trace symbol, T_r, denotes a sum over all possible configurations of atoms in the sample system. The logarithm of Z, called the free energy, $F(T)$, contains information about the average energy, $<E(T)>$, and also the entropy, $S(T)$, which is the logarithm of the number of configurations contributing to the ensemble at T;

$$-K_B T \ln Z = F(T) = \; < E(T) > -S(T) \tag{3.2}$$

Boltzmann-weighted ensemble averages can be easily expressed in terms of derivatives of F. and the rate of change of the energy with respect to the control parameter, T, is related to the size of typical variations in the energy by

$$C(T) = \frac{d\langle E(T)\rangle}{dT} \tag{3.3}$$

In statistical mechanics, *C(T)* is called the specific heat. A large value of *C* signals a change in the state of order of a system, and can be used in the optimization context to indicate that freezing has begun and hence that very slow cooling is required. It can also be used to determine the entropy by the thermodynamic relation

Of the approaches to handle the analog placement problem, SA is used as the search engine based on a cost function that includes area, wire length, and other penalties. Although an SA-based approach performs well in the placement problem, to ensure the convergence to the optimum, much effort is required for the problem definitions and the parameter tune-up, such as move types, neighbor structures, and annealing schedules.

3.2 Genetic Algorithm

A genetic algorithm (GA) [35] is a search technique used in computing to find exact or approximate solutions to optimization or search problems. Genetic algorithms are categorized as global search heuristics and a particular class of evolutionary algorithms (EA) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. GA is implemented in a computer simulation in which a population of abstract representations (called

chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of each individual in the population is evaluated. Multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

A typical genetic algorithm requires:

1) a genetic representation of the solution domain;

2) a fitness function to evaluate the solution domain.

A standard representation of the solution is an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and

graph-form representations are explored in evolutionary programming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of the solution might be an array of bits, where each bit represents a different object, and the value of the bit (*0* or *1*) represents whether or not the object is in the knapsack. Not every representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Once we have the genetic representation and the fitness function defined, genetic algorithm proceeds to randomly initialize a population of solutions and then improve it through repetitive application of mutation, crossover, inversion and the selection operators.

Zhang *et al* [32] proposed a genetic algorithm associated with certain simulated annealing concepts to handle the analog placement problem. The process is described as follows. First, the geometry information and the net-list of the cells are inputted. Then the first population is randomly initialized and several random placement states are initialized to setup the initial temperature *T*. The next step is to evaluate the cost with the aid of the cell-slide process which is to place the cells adjacent and no overlap.

The iteration based on the crossover-rate begins to do crossover to generate one offspring and do inversion on the generated offspring based on inversion-rate. Then the best N members are chosen among the former members and newly generated offspring, and they are set as the new generation. For each of $N/2$ members in the new generation, mutation on the clone of one member is done based on the mutation-rate. The iteration continues until the temperature reaches the stop criterion. At last, it outputs the best results. The flow of the algorithm is shown in Figure 3.1. It is reported that this proposed algorithm along with the optimized parameters, with high computation efficiency, can generate high-quality placement of the cells.

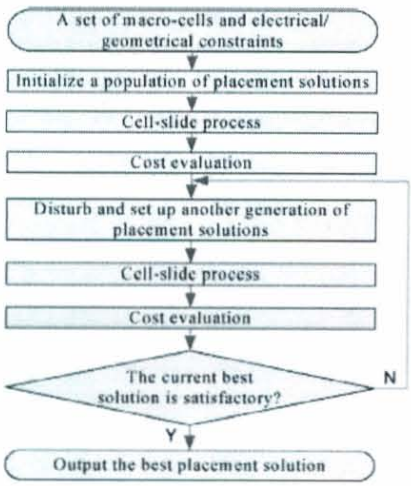


Figure 3.1 Overflow of the GA-based analog placement algorithm

3.3 Neutral Network

In this section, a method to handle the placement problem using artificial neutral network is detailed to compare with the previous two search engines. An artificial neural network (ANN) [36] is a mathematical model or computational model that tries to simulate the structure and/or functional aspects of biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionism approach to computation. In most cases, an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Neural networks are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data. In the following sections, I will introduce the details of the neutral network and its application to solve the analog placement problem.

3.3.1 Introduction to Neutral Network

Although many schemes have been proposed for the analog layout automation, a majority of the current placement algorithms deploy simulated annealing [18] or evolutionary algorithms [35], whose performance heavily relies on the applied cooling schedule or the construction of perturbation/mutation operators.

The development of artificial neural networks provides us more choices to solve the placement problem. Thus far, some neural networks have been adapted for this

purpose. Kita *et al.* [36] proposed a Hopfield network model to handle the placement problem by introducing several schemes (e.g., interconnection modification and weight-factor tuning) for the energy function. This neural network can generate proper placement that has sufficiently good quality. However, this work only considered the basic area and wire requirements normally demanded by digital VLSI circuits. Gloria *et al.* presented a neural model called Boltzmann machine to solve the block placement problem on parallel machines [37]. This work is particularly suited for the low-cost massively parallel implementation in order to reduce execution time at the price of processor and memory resources.

Fang *et al.* proposed a mean-field neural network model for quadratic assignment [38]. According to their theoretical study, the mean-field neural network is able to repeat the annealing flow at particular temperature T_c so that the optimal solutions can be searched with less execution time. The analysis shows the iteration numbers should be proportional to T_c / (T_c-T), which means more iteration is required to reach equilibrium around T_c than anywhere else. Furthermore, Unaltuna and Pitchumani designed a normalized neural network that is able to find optimal solution with lower cost compared to Hopfield network [39]. In addition, they provided a good scheme to determine the temperature T_c.

To the best of my knowledge, applying neural network to the analog placement problem has yet to be investigated. Therefore, in this thesis, I extend the previous work by focusing on the analog constraints.

3.3.2 Analog Placement Problem Modeling

Assume a list of cells $\{C_1, C_2, C_3, ..., C_n\}$ are to be placed on a plane whose width and length are W and H, respectively. So we design a mean-field neural network with $n \times W \times H$ neurons. Each neuron has a binary value output. For the problem modeling, we define a set of three-dimensional binary array $V = \{ O_{C_i, x_i, y_i} \}$ to represent neurons. The network architecture is shown in Figure 3.2. If we place one cell C_i on the plane and the coordinates of the left-bottom vertex is (x_i, y_i), then O_{C_i, x_i, y_i} will be *1*. Otherwise, the output will be *0*. We update the state of each neuron following the mean-field algorithm described in section 3.3.3 until the network reaches thermal equilibrium. Based on the final state of neurons, we can obtain an optimal placement.

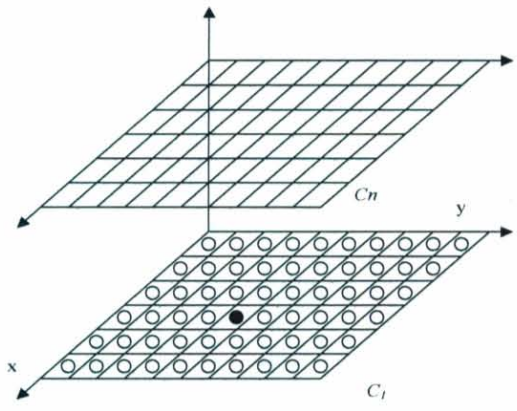


Figure 3.2 Network Architecture

To obtain a valid placement, we need to consider the following constraints: (1) each cell should be placed at one and only one particular position on the plane; (2) any two cells should not overlap with each other; (3) any cells should not be placed over the boundary of the plane; (4) to extend the use of the network, we take the symmetry and proximity constraints into account.

After applying the aforementioned constraints, we can obtain a valid placement. The goal of the analog placement problem is to locate the cells within the plane whose size is $W \times H$. And the total wire interconnection should be minimized.

3.3.3 Algorithm Description

To solve the analog placement problem, we design the mean-field neural network algorithm with the aid of several parameters as shown below.

1) Energy function

a) Basic energy function

In this section, we define some parameters to indicate the characteristics of cells. We create an $n \times n$ matrix L to include the interconnections between each two cells. $L(C_i, C_j)$ specifies the number of interconnections between cells C_i and C_j (i.e., how

many ports are connected between these cells). The Euclidean distance value between the two cells is defined as $d(C_i, C_j)$ by calculating the distance between the cell centers in the algorithm.

Based on the variables defined above, we construct the basic energy function as follows:

$$E_1 = \sum_{C_i} \sum_{x_i} \sum_{y_i} \sum_{C_j} \sum_{x_j} \sum_{y_j} L(C_i, C_j) \\ d(C_i, C_j) O_{C_i, x_i, y_i} \cdot O_{C_j, x_j, y_j} , \qquad (3.4)$$

where O stands for the output of neuron. We can see this basic energy function decreases when two cells are placed with less distance.

b) Non-overlap constraints

The non-overlap constraint means no cell is allowed to overlap with another in the final placement. As $w(C_i)$ and $h(C_i)$ are used to indicate width and height of cell C_i, we can easily set up the following constraint for all the cells:

$$\sum_{C_j, j \neq i} \sum_{u=-(w(C_j)-1)}^{w(C_i)-1} \sum_{t=-(h(C_j)-1)}^{h(C_i)-1} O_{C_i, x_i, y_i} \cdot O_{C_j, x_i+u, y_i+t} = 0 , \qquad (3.5)$$

Therefore, we have the second energy function item as a penalty component shown in (3.9):

$$E_2 = \sum_{C_i}\sum_{x_j}\sum_{y_j}\sum_{C_j, j\neq i}\sum_{u=-(w(C_j)-1)}^{w(C_j)-1}\sum_{t=-(h(C_j)-1)}^{h(C_j)-1} O_{C_i,x_i,y_i}\cdot O_{C_j,x_i+u,y_i+t}\,. \quad (3.6)$$

c) Symmetry constraints

Symmetry constraints involve two types of cases: symmetric pair and self-symmetric cells. Cells in one symmetric pair should be placed on the opposite sides with respect to symmetry axis, whereas the self-symmetric cell should be placed along the symmetry axis. In this work, we only consider the horizontal symmetry situation. We use an $n\times n$ matrix S to indicate the symmetry relationship between the cells. If cells C_i and C_j are a symmetric pair, then $S(C_i,C_j) = S(C_j,C_i) =1$, otherwise, $S(C_i,C_j) = S(C_j,C_i)=0$. If cell C_i is self-symmetric, then $S(C_i,C_i) =1$, otherwise, $S(C_i,C_i) =0$. Assume we have p symmetric pairs and q self-symmetric cells. We use p_1 and p_2 to denote any symmetric cells. Now we consider a symmetric pair $P_k =(C_i,C_j)$ whose two cells have the same Y coordinates and are placed on the opposite sides with respect to the same symmetry axis.

We assume the center X coordinates of one symmetric pair and one self-symmetric are X_p and X_q, respectively. And X_s stands for the corresponding symmetry coordinates.

$$X_{s_1} = X_{P_1} + w_{P_1}/2 + (X_{P_2} - X_{P_1})/2,$$

$$X_{s_1} = X_{q_1} + w_{q_1}/2,$$

$$E_3 = \sum_{C_i}\sum_{x_i}\sum_{y_i}\sum_{C_j}\sum_{x_j}\sum_{y_j} S(C_i, C_j)$$
$$\Big(\sum_{s_p, s_p \neq s_i} |(X_{s_p} - X_{s_i})| + \sum_{s_q, s_q \neq s_i} |(X_{s_q} - X_{s_i})| \Big) \cdot \tag{3.7}$$
$$+(|y_{C_i} - y_{C_j}|) O_{C_i, x_i, y_i} \cdot O_{C_j, x_j, y_j}$$

d) Proximity constraints

The proximity constraints limit cells to a connected placement so that the devices can share a common substrate/well region or guard ring in order to decrease parasitic effect. To represent proximity relationship, we use an $n \times n$ matrix K. If cells C_i and C_j have proximity relationship, then $K(C_i, C_j) = 1$ and $K(C_j, C_i) = 1$, otherwise, $K(C_i, C_j) = 0$. So we can have the energy function E_4 to apply the proximity constraints to the placement:

$$E_4 = \sum_{C_i}\sum_{x_i}\sum_{y_i}\sum_{C_j}\sum_{x_j}\sum_{y_j} K(C_i, C_j) d(C_i, C_j) O_{C_i, x_i, y_i} \cdot O_{C_j, x_j, y_j} \quad . \tag{3.8}$$

e) Boundary constraints

As the plane is fixed, we need to set up boundary constraints to place cells only inside the plane. For every cell, we must keep $0 < x_i < W - w(C_i)$ and $0 < y_i < H - h(C_i)$. So we can have a fifth energy function as follows:

$$E_5 = \sum_{C_i}\sum_{x_i}\sum_{y_i} \sum_{r=(W-w(C_i)-1)}^{W-1} \sum_{v=(H-h(C_i)-1)}^{H-1} O_{C_i,x_i,y_i} \cdot O_{C_i,r,v} . \qquad (3.9)$$

We can obtain the entire energy function of the mean-field neural network for the analog placement problem in the following:

$$E = \alpha_1 \cdot E_1 + \alpha_2 \cdot E_2 + \alpha_3 \cdot E_3 + \alpha_4 \cdot E_4 + \alpha_5 \cdot E_5, \qquad (3.10)$$

where α_1, α_2, α_3, α_4, and α_5 are weight factors. The selection of the weight factors may affect the optimization process of the neural network. When selecting large values, the network has fast convergence speed but maybe sacrifice the search quality, i.e., the optimization is premature and a non-optimal solution is obtained. However, when we select small values, the network can obtain a better solution but with long computing latency. We tune the factors after computation experiment and follow the standard of choosing the factors describe in [36].

Furthermore, we can obtain the *mean field* by calculating the first-order partial derivative of E in terms of O_{C_i,x_i,y_i} to update the state of the neurons.

$$\phi_{C_i,x_i,y_i} = \frac{dE}{d(\sum_{C_i}\sum_{x_i}\sum_{y_i} O_{C_i,x_i,y_i})} . \qquad (3.11)$$

2) Plane constraint

For a general placement problem, we need to set up a plane constraint. This constraint means that each cell must be placed within one panel (i.e., neuron) on the plane in Figure 3.2. Therefore, in the three-dimensional matrix of Figure 3.2, there is only one neuron, whose output is *1*, on one plane formed by *X* and *Y* dimensions. In contrast, for the Hopfield network, we need to add a penalty item to the energy function in order to apply this constraint. Generally speaking, it is difficult to determine the penalty item in a neural network to solve the analog placement problem. Therefore, in this thesis, we employ the mean-field neural network [7] and normalization of neurons to apply this constraint.

The output state vector O_{C_i,x_i,y_i} of each neuron can be treated as a unit in the random equilibrium disturbance. So we can assume that the probability of locating the coordinates of left-bottom corner of cell C_i accords with the Boltzmann distribution as shown in (3.12),

$$o_{C_i,x_i,y_i} \approx \exp(-\phi_{C_i,x_i,y_i} / T), \qquad (3.12)$$

where ϕ_{C_i,x_i,y_i} is the mean field as given in (3.11).

When ϕ_{C_i,x_i,y_i} increases, the probability O_{C_i,x_i,y_i}, which is the output of neuron,

will decrease. When output O_{C_i,x_i,y_i} decreases to zero, the left-bottom corner of cell C_i will not be assigned to the position of (x_i, y_i). To obtain a practical probability, we apply normalization of the neuron output by dividing the sum of the value of all neurons as shown in (3.13):

$$O_{C_i,x_i,y_i} = \exp(-\phi_{C_i,x_i,y_i} / T) / \sum_{C_j}\sum_{x_j}\sum_{y_j} \exp(-\phi_{C_j,x_j,y_j} / T). \tag{3.13}$$

The equations mentioned above would guarantee that cells can be placed following the plane constraint. Obviously, this normalization scheme is beneficial as it removes the requirement of the penalty item. The normalization scheme and the energy function can guarantee to obtain a valid solution to the placement problem.

3) Critical temperature

As mentioned in Section 3.3.1, one of the advantages of the mean-field neural network is that it may achieve the optimal solution at particular temperature without decreasing the temperature [38] [40]. This temperature is called *critical temperature* where the bulk of the optimization occurs. To obtain the critical temperature, we can assume that in the initial state, the occupation probability of each cell in each grid position is the same, $1/\sum_{C_j}(W \cdot H)$. So we can determine the critical temperature using (3.14):

$$T_c = \max_{C_i, x_i, y_i} \left\{ \phi_{C_i, x_i, y_i} / \ln(\Sigma_{C_j} (W \cdot H)) \right\} \qquad (3.14)$$

And we choose T_c as the critical temperature and start the annealing process from that temperature.

4) Algorithm flow

The annealing flow for solving the analog placement problem is shown in Fig 3.3.

Begin

1 Randomly assign values to neurons and generate an initial placement;

2 Calculate the normalization of all the neurons;

3 Calculate T_C using (3.14) as the initial temperature;

4 Calculate the energy $E_{old}=E$ using (3.10);

5 **Do**

6 Randomly choose one cell C_i;

7 Calculate the mean field ϕ_{C_i,x_i,y_i} for all x and all y with (3.11). Add hill-climbing item to (3.11) to avoid local minima;

8 Update the output O_{C_i,x_i,y_i} of the neurons for all x and all y using (3.12);

9 Calculate normalization of O_{C_i,x_i,y_i} for all x and all y using (3.13);

10 Calculate E_{new} using (3.10);

11 Calculate deltaE= E_{new} - E_{old};

12 **While** (*deltaE* is not small enough)

End

Figure 3.3 ANN placement algorithm

After establishing a mean field neural network and inputting a circuit netlist, we first randomly assign each neuron with a value, do the normalization and calculate the critical temperature as shown in Lines 1-3 of Figure 3.3. Then we randomly choose one

cell and update the neurons on the plane corresponding to this cell using (3.12) and calculate the energy change (in Line 11) after update. When updating the neurons, we first need to calculate the ϕ_{C_i,x_i,y_i}. Here local minima means all the output of the network is close to zero and we cannot obtain a better solution. To escape from this situation, we need to add a value (called *hill-climbing item* as shown in Line 7 of Figure 3.3). This update process is repeated until the network reaches thermal equilibrium. And the final output of the network gives an optimal solution to the placement problem.

The complexity to calculate the mean field ϕ_{C_i,x_i,y_i} is *O(N)*, where *N* is the number of neurons. To calculate the energy, we only need to calculate *deltaE* (shown in Line 11 of Figure 3.3) and the complexity is $O(N^2/n)$, where *n* is the number of cells needed to be placed. We need *O(n)* iteration to reach the thermal equilibrium state. So the total complexity of the algorithm is $O(N^2)$.

3.3.4 Neural Network Method Results

We implemented this neural network algorithm in Java and tested it on a 2GHz PC. To compare with other work, we employ two test circuits with added symmetry and proximity properties. The first test circuit has 20 cells from [37]. Reference [37] provides a solution known to be the best optimal to this circuit. The second test circuit that we used in this study has 25 cells provided in [36]. Both circuits are input together with interconnection, symmetry, and proximity matrixes to initialize distinct

properties.

We can first see a simple case of 4 cells whose width and length is *2×2* need to be placed at a *10×10* plane. The result is shown in Figure 3.4. We have to build a *10 × 10×4* neural network first. We can see that the four matrixes denote the location of the four cells. The maximum possibility of *1* in matrices *1, 2, 3, 4* are in the position (*9, 2*), (*9, 4*), (*7, 4*) and (*7, 2*) respectively. So we obtain the final placement with *4* cells placed closely and the cost is minimized.

```
0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119
0.0107 0.0092 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0107 1.0000 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119

0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119
0.0107 0.0092 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0107 0.0092 0.0094 1.0000 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119

0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119
0.0107 0.0092 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 1.0000 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0107 0.0092 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119

0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119
0.0107 0.0092 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 1.0000 0.0096 0.0095 0.0095 0.0095 0.0095 0.0096 0.0094 0.0109
0.0109 0.0094 0.0096 0.0096 0.0096 0.0096 0.0096 0.0096 0.0094 0.0109
0.0107 0.0092 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.0092 0.0107
0.0119 0.0107 0.0109 0.0109 0.0109 0.0109 0.0109 0.0109 0.0107 0.0119
```

Figure 3.4 The result of a simple 4 cells placement problem

We compare our work with simulated annealing (*SA*), genetic algorithm (*GA*) [35], Hopfield network [36], and Boltzmann machine [37]. Our algorithm presented

in this chapter is marked as *MF*. Since the previous work did not consider the analog placement constraints, we added some related weight items to the Hopfield and Boltzmann machine methods. For simple implementations of *SA* and *GA*, we added a checking function to exclude invalid placements. In Table 3.1, we show the comparison results from 50 times execution, and we pick the best results, the worst results, and the average results. From Table 3.1, we can see that our *MF* algorithm obtained better placement with cost improvement of 9%, 15%, and 13% over *SA*, *GA*, and Hopfield methods, respectively, for the best results of Circuit1. In terms of execution time, the *MF* algorithm ran 21%, 18%, and 24% faster than *SA*, *GA*, and Hopfield methods, respectively. In contrast, for the average results of Circuit1, our proposed algorithm achieved better performance in cost by 16%, 16%, and 17% and less execution time by 14%, 17%, and 20% compared to *SA*, *GA*, and Hopfield methods. Compared to the Boltzmann machine method, *MF* had higher cost of 5%, 1%, and 1% for the best, the worst and the average results, respectively. However, our proposed algorithm reduced the execution time by 5%, 8%, and 6%, respectively. One final placement result of Circuit1 is depicted in Figure 3.5.

For the best results of Circuit 2, our proposed algorithm achieved better performance in cost by 8%, 8%, and 10% and reduced the execution time by 7%, 7%, and 15% compared to *SA*, *GA*, and Hopfield methods, respectively. Although *MF* worked marginally worse (by 1%) in terms of cost than Boltzmann machine method, the former algorithm ran a bit faster (by 4%) than the latter one.

Table 3.1 Neural network algorithm circuits results

Circuit		*SA*	*GA*	*Hopfield*	*Boltzmann*	*MF*
Circuit1	Best Cost	156	167	164	134	142
	Time (sec.)	78	75	81	64	61
	Worst Cost	201	204	197	168	170
	Time (sec.)	64	55	60	59	54
	Average Cost	187	186	189	154	156
	Time (sec.)	72	70	73	62	58
Circuit2	Best Cost	837	827	845	752	763
	Time (sec.)	164	162	178	159	152
	Worst Cost	901	887	883	832	840
	Time (sec.)	186	189	193	167	164
	Average Cost	879	867	865	802	800
	Time (sec.)	175	177	173	154	150

In summary, our mean-field neural network algorithm is able to achieve better performance in both cost and the execution time than *SA*, *GA*, and Hopfield methods. Compared to Boltzmann machine method, our proposed algorithm can reduce the execution time but experience marginal performance loss in cost.

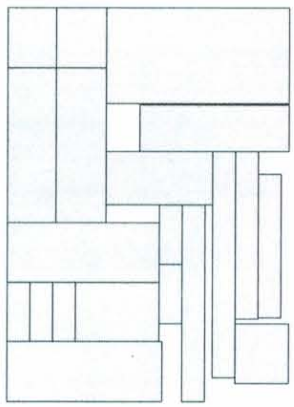


Figure 3.5 Final placement of Circuit1 using *MF*

3.3.5 Neural Network Method Summary

We first discussed the modeling of the problem in terms of neural network and then presented a detailed optimization flow in particular on how to manage the multiple constraints for analog placement. Our experimental results show the efficiency of the proposed algorithm compared to some other stochastic approaches. However, one big issue for this problem is that it cannot guarantee to obtain a valid placement all the time. Thus, a post-processing step is needed to check whether the result is correct and fix the place plane if it is invalid. This drawback motivates me to develop a better algorithm to resolve the problem for the general analog placement design.

3.4 Summary

Most of the work of the analog placement design uses simulated annealing or genetic algorithm. An artificial neural network based approach is implemented to handle the analog placement problem. However, this algorithm cannot always guarantee to obtain a valid placement. And the plane size has to be fixed at the beginning of optimization. Therefore, in the following chapters I am focused on

developing an efficient topologic placement algorithm to resolve this problem by nature. As most of the previous work used simulated annealing as the search engine, to maintain a fair comparison, we will use the same scheme as the search engine for the proposed TCG-based algorithm in this thesis.

Chapter 4 TCG-based Method to Handle the Symmetry Constraints

In this chapter, I will introduce a strategy for the placement problem to handle the symmetry constraints with multiple groups. A set of symmetric-feasible definitions for TCG is first presented in Section 4.1. Then the packing flow will be detailed and the perturb operations will be discussed in Sections 4.2 and 4.3. Finally I will show the experimental results and draw the conclusions in Section 4.4.

4.1 Symmetric-Feasible TCG

In this section, we first present one definition for TCG symmetric-feasible conditions. Then the correctness of the conditions is proved. By using the proposed conditions, one can verify the feasibility of symmetric placements without packing in advance. Without loss of generality, the analysis in this thesis is only focused on the situation where the symmetry axis is vertical.

Let (G_h, G_v) be the TCG representation of a placement containing two symmetry groups Γ and Φ. For $(a, a') \in \Gamma$, if $a \neq a'$, then (a, a') is a symmetric pair consisting of

two distinct cells a and a'; and if $a=a'$, then a is a self-symmetric cell. For the multi-group symmetry situation, we can define the following conditions.

Definition 1: For $\forall(a, a') \in \Gamma$, $\forall(b, b') \in \Gamma$, $\forall(c, c') \in \Phi$, and $\forall(d, d') \in \Phi$, a TCG representation is symmetric-feasible if the following four conditions are satisfied.

For intra-group of Γ (the same for Φ)

$$\text{in } G_h : a \vdash b \langle\neq\rangle a' \vdash b', \tag{4.1}$$

$$\text{in } G_v : a \perp b \langle\neq\rangle b' \perp a'; \tag{4.2}$$

For inter-group between Γ and Φ

$$\text{in } G_h : a \vdash c \text{ and } a' \vdash c' \langle\neq\rangle d \vdash b \text{ and } d' \vdash b', \tag{4.3}$$

$$\text{in } G_v : a \perp c \langle\neq\rangle c' \perp a'; \tag{4.4}$$

where symbol $\vdash$ represents its left operand is topologically on the left of its right operand, symbol $\perp$ represents its left operand is topologically below its right operand, and symbol $\langle\neq\rangle$ denotes that the two cases before and after this symbol cannot simultaneously appear in the same TCG. For the intra-group conditions, (4.1) and (4.2) guarantee the symmetric feasibility of symmetric cells within one group. In contrast, the inter-group conditions (4.3) and (4.4) are deployed to coordinate the relative location of symmetric cells among multiple symmetry groups.

Lemma 1: Any symmetric placement containing multiple symmetry groups can be represented by a symmetric-feasible TCG.

Proof: First, we consider the relationship between symmetry groups since the intra-group relationship can be considered as a special case of multiple symmetry groups.

For any two symmetric pairs $(a,\ a') \in \Gamma$ and $(b,\ b') \in \Gamma$, and another two symmetric pairs $(c,\ c') \in \Phi$ and $(d,\ d') \in \Phi$, to simplify the analysis, we first assume two pairs within one symmetry group are located vertically, that is, $a \perp b$, $a' \perp b'$, $a \vdash b'$, and $b \vdash a'$ in Γ, $c \perp d$, $c' \perp d'$, $c \vdash d'$, and $d \vdash c'$ in Φ. Thus, there are four typical relative positions as shown in Figure 4.1. Cases (I) and (II) show the situation where the two symmetry groups are placed without interference, whereas cases (III) and (IV) indicate the situation where the two symmetry groups interfere with each other.

Case (I): $(a,\ a')$ and $(b,\ b')$ are placed at the left of $(c,\ c')$ and $(d,\ d')$. In other words, group-Γ is placed at the left of group-Φ as depicted in Fig. 4.1(I). The relationships of the cells in G_h are:

$$a \vdash c,\ a' \vdash c',\ a \vdash c',\ a' \vdash c,$$

$$b \vdash d,\ b \vdash d',\ b' \vdash d',\ b' \vdash d,$$

$$a \vdash d,\ a' \vdash d',\ a \vdash d',\ a' \vdash d,$$

$$b \vdash c,\ b \vdash c',\ b' \vdash c' \text{ and } b' \vdash c.$$

And the relationships in the G_v are:

$$a \perp b,\ a' \perp b', \qquad\qquad c \perp d,\ c' \perp d'.$$

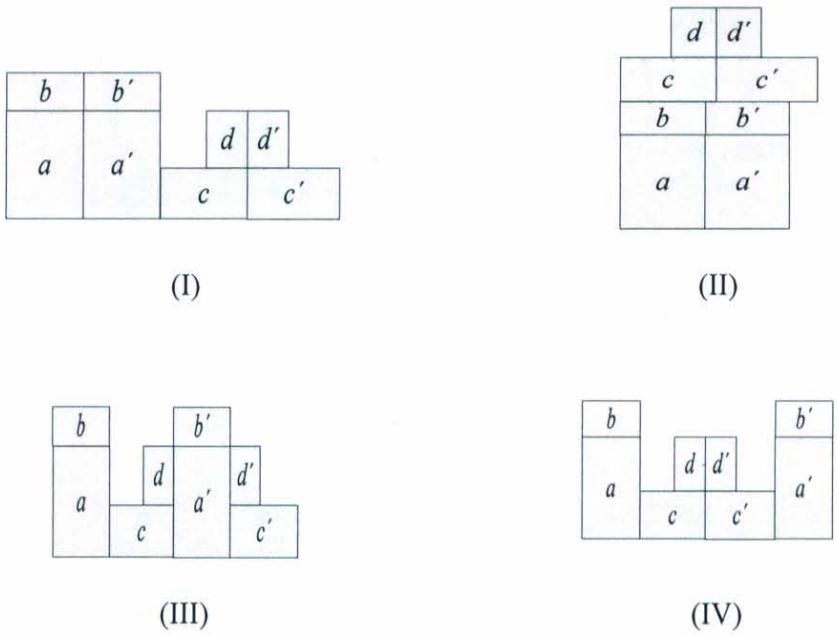


Figure 4.1 (I) group Γ is placed at the left of group Φ; (II) Γ is placed below Φ; (III) Γ and Φ are intermingled; (IV) Φ is placed within Γ

Obviously, there is no violation above in terms of the symmetric-feasibility conditions (4.1)-(4.4).

Case (II): as group-Γ is placed above group-Φ, and the relationships between the cells are list as follows:

$$a \perp c,\ a' \perp c',\ a \perp c',\ a' \perp c,$$

$$b \perp d,\ b \perp d',\ b' \perp d',\ b' \perp d,$$

$$a \perp d,\ a' \perp d',\ a \perp d',\ a' \perp d,$$

$$b \perp c,\ b \perp c',\ b' \perp c' \text{ and } b' \perp c$$

And the relationships in the G_h are:

$$a \vdash b,\ a' \vdash b', \qquad\qquad c \vdash d,\ c' \vdash d'.$$

we can prove it in the same way as case (I).

Case (III): This case shows that group-Γ and group-Φ are placed with interference. The relationships of the cells in G_h are:

$$a \vdash c,\ a \vdash d,\ a \vdash c',\ a \vdash b',$$

$$c \vdash a',\ d \vdash a',\ c \vdash b',\ d \vdash b',$$

$$a' \vdash c',\ a' \vdash d',\ b' \vdash c',\ b' \vdash d',$$

And in G_v we can have the relationships:

$$a \perp b,\ a' \perp b', \qquad\qquad c \perp d,\ c' \perp d'.$$

The relationships listed above do not violate the symmetry conditions of (4.1)-(4.4).

Case (IV): The relationships of the cells in G_h are:

$$a \vdash c,\ a \vdash d,\ a \vdash c',\ a \vdash b',$$

$$c \vdash a',\ c \vdash d',\ c \vdash b',$$

$$b \vdash c,\ b \vdash d,\ b \vdash a',$$

$$d \vdash b',\ d \vdash c',\ d \vdash a',$$

we can have the relationships in G_v:

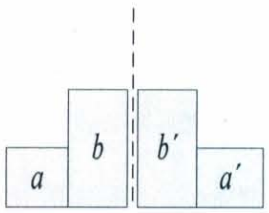
$$a \perp b,\ a' \perp b', \qquad\qquad c \perp d,\ c' \perp d'.$$

We can see the same result holds for this case, where group-Φ is placed inside

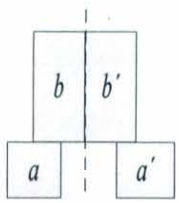
group-Γ.

Besides, in a similar manner we can analyze the other situations, e.g., the symmetric pairs within one symmetry group are located horizontally or any symmetric pair is replaced by one self-symmetric cell. Therefore, we can derive that the symmetric-feasibility conditions (4.3)-(4.4) will always hold for any analog placements with multiple symmetry groups.

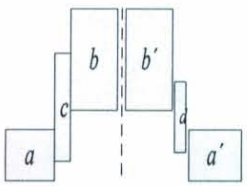
As for the intra-group conditions, we can consider one single symmetry group, say Γ. Assume $(a, a') \in \Gamma$ and $(b, b') \in \Gamma$. There are three typical placements as shown in Figure 4.2, where cells c and d may be any cells. If $a = a'$ (or $b = b'$), then a (or b) is a self-symmetric cell.



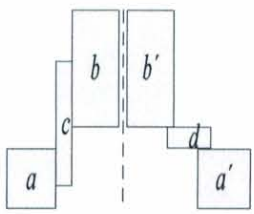
(I)



(II)



(III.a)



(III.b)

Figure 4.2 (I) (b, b') are placed within (a, a'); (II) (a, a') are placed at the bottom of (b, b'); (III.a) $a \vdash b$, $b' \vdash a'$, $a \vdash c$, $c \vdash b$, $b' \vdash d$, and $d \vdash a'$; (III.b) $a \vdash b$, $a' \perp b'$, $a \vdash c$, $c \vdash b$, $d \perp b'$, and $a' \perp d$.

In case (I) of Figure 4.2, (b, b') sit within (a, a') along the horizontal direction, and their vertical projections overlap. In this case, no relationship in G_v exists and the following relationships in G_h hold:

$$a \vdash b \text{ and } b' \vdash a',$$

which satisfy (4.1) in Definition 1. In case (II), (b, b') are placed on the top of (a, a') and there are relationships of $a \perp b$ and $a' \perp b'$, which accord with (4.2) in Definition 1. In case (III), we can categorize it into two situations. As shown in situation (III.a), cells c and d are placed within the two pairs, and their vertical projections overlap. Thus, this situation is the same as case (I). As for situation (III.b), we have $a \vdash b$ (due to $a \vdash c$ and $c \vdash b$) on the left side of the symmetry axis, whereas we have $a' \perp b'$ (due to $a' \perp d$ and $d \perp b'$) on the right side. And these geometry relationships in the graphs have no violation from (4.1) and (4.2).

Therefore, we can conclude that any symmetric placement with multiple symmetry groups can be represented by a symmetric-feasible TCG. ∎

(Note: symbol ∎ stands for an end of proof)

4.2 Symmetric Packing

In this section, I will describe the flow of the packing scheme for TCG. To construct placement of a symmetric-feasible TCG, the conventional TCG representation has been adapted by introducing a dummy axis cell to denote the

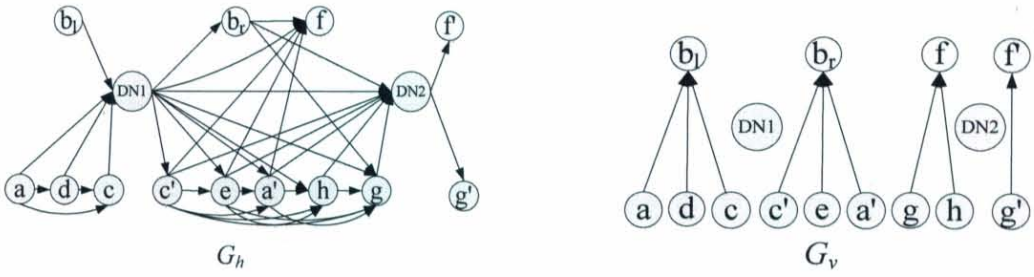
symmetry axis of each symmetry group. One example is shown in Figure 4.3, where there are two symmetry groups. Group-1 has two symmetric pairs (i.e., (a, a') and (c, c')) and one self-symmetric cell b, whereas group-2 has two symmetric pairs (i.e., (g, g') and (f, f')). Cells d, e, and h are asymmetric. For the two symmetry groups, we add dummy axis node *DN1* for group-1 and *DN2* for group-2. In the packing and perturbation process, we employ the dummy nodes as barriers to separate the TCG into different parts with respect to different symmetric axes. To make it convenient to operate on self-symmetric cells, we divide any self-symmetric cell into a pair. In Figure 4.3 (a)-(c), b_l and b_r stand for the two halves (i.e., the left and right part) derived from the self-symmetric cell of b. For any asymmetric cells that overlap with a symmetry axis, we consider them to be constrained by a horizontal relationship with the symmetric axis. Figure 4.3 (d) shows the corresponding symmetric placement. To make the graph more comfortable to read, we only keep the edges between the cells in each divided area.

I have developed a packing scheme as listed in Figure 4.4. Below the packing process is explained based on the example depicted in Figure 4.3. The proposed flow is composed of four major steps:

Step 1: packing preparation (Lines 1-4).

First we obtain the topological order (i.e., the β-sequence of the corresponding SP) that represents the packing sequence of the TCG. Then we use the dummy axis nodes as barriers to divide the entire sequence into multiple sub-sequences as shown in

Figure 4.6. The purpose of this operation is to keep symmetry requirements void within each sub-sequence;

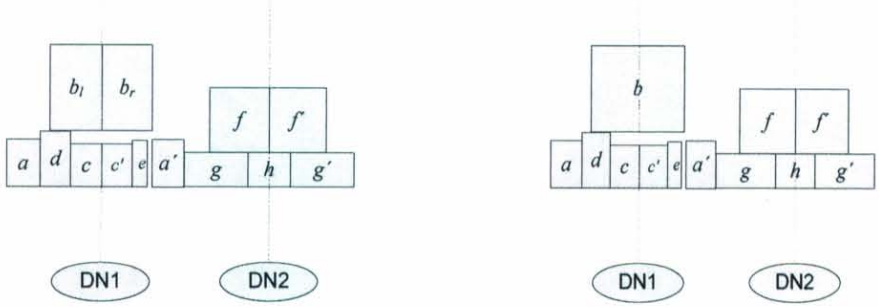


(a)

(b_l, a, d, c, DN1, b_r, c', e, a', f, g, h, DN2, f', g')

(a, d, c, b_l, DN1, c', e, a', b_r, g, h, f, DN2, g', f')

(b)



(c) (d)

Figure 4.3 (a) TCG; (b) corresponding SP; (c) symmetric placement with separate self-symmetric halves; (d) final symmetric placement

Step 2: initial packing (Lines 5-7).

Here we borrow the concept of contour adopted in TCG-S [16] to operate on a step called *initial packing* for compactly placing each sub-sequence. Two lists (i.e., C_h and C_v for horizontal and vertical contours, respectively) are initialized empty and updated once each new cell is placed. The cells are processed one by one according to the topological order and a current cell is always placed horizontally/vertically adjacent to the cells that are already on the contour list. The packing is started from the most left bottom coordinates of (0, 0). For instance, in Figure 4.6, to pack each sub-sequence, each time we only consider the cells in the shadow region. That is to say, we first pack sub-sequence (a, d, c, b_l), then the second sub-sequence (c', e, a', b_r, g, h, f), and last the third sub-sequence (g', f').

The contour scheme is based on the topological order and both C_h and C_v keep updated. The basic idea is to process the cells following the sequence defined in the topological order and locate the current cell to a corner fixed by the previously placed cells in C_h and C_v according to the geometric relationship defined in G_h and G_v. As well, we keep the information of valid segment value with the cells. We use x_i (x_i') and y_i (y_i') to denote the X and Y coordinates of the left-bottom (right-top) corner of cell c_i. Recall that C_h (C_v) is a list of cell c_i's for which there exists no module c_j with $y_j > y_j'$ ($x_j > x_j'$). C_h (C_v) consists of the cells along the top (right) boundary of a placement. We can keep the cell c_i's in C_h (C_v) in a balanced binary search tree T_h (T_v) in the increasing order according to their right (top) boundaries. For easier presentation, we add a *dummy cell* c_l (c_b) to C_h (C_v) to denote the left (bottom) boundary cell of a placement. We have $c_l \vdash c_i$ and $c_b \perp c_i$, for all c_i in the placement.

Let $(x_l', y_l') = (0, \infty)$, and $(x_b', y_b') = (\infty, 0)$. $C_h(C_v)$ consists of c_l (c_b) initially, and so does the corresponding $T_h(T_v)$. To pack a cell c_i in the topological order, we traverse the cell c_k's in $T_h(T_v)$ from its root, and go to the right child if $c_k \vdash c_i$ ($c_k \perp c_i$) and the left child if $c_i \perp c_k$ ($c_i \vdash c_k$). The process is repeated for the newly encountered cell until a leaf node is met. Then, c_i is connected to the leaf node, and $x_i = x_p'$ ($y_i = y_p'$), where c_p is the last cell with $c_p \vdash c_i$ ($c_p \perp c_i$) in the path. After c_i is inserted into $T_h(T_v)$, every predecessor c_l with $x_l' < x_i'$ ($y_l' < y_i'$) in $T_h(T_v)$ is deleted since c_l is no longer on the contour. The ordering of nodes in $T_h(T_v)$ can be obtained by depth-first search. This process repeats for all cells in the topological order. Thus, we have $W = x_v'$ ($H = y_v'$) if c_v is the cell in the resulting $T_h(T_v)$ with the largest value, where W (H) denotes the width (height) of the placement.

Begin

1 Construct the topological order of the TCG;

2 Generate the packing sequence of *packSeq*, which contains only the symmetric cells;

3 Consider the dummy axis cells as barriers to separate *packSeq* into sub-sequences;

4 Create a *list* made up of the sub-sequences;

5 Do *initial packing* for the first sub-sequence and calculate the axis position X_{axis};

6 **For** (the *list* is not empty)

7 Do *initial packing* for the second sub-sequence;

8 Follow *packSeq* one by one. Compare the Y coordinates of each two cells in one symmetric pair such as (a, a'), and make $Y_a = Y_{a'} = max(Y_a, Y_{a'})$ and shift the packing symmetric cells that have vertical relationship with the shifted cells;

9 Follow *packSeq* one by one, for every two cells in one symmetric pair such as (a, a'); calculate $\Delta X_a = |X_a - X_{axis}|$ and $\Delta X_{a'} = |X_{a'} - X_{axis}|$. Then shift one symmetric cell to make $\Delta X_a = \Delta X_{a'} = max\ (\Delta X_a, \Delta X_{a'})$. Also tune-up the corresponding cells with same ΔX_a;

10 Do final packing for the first and second sub-sequences;

11 Remove the first sub-sequence from the sub-sequence *list*;

12 **End for**

13 Consider all symmetric cells as preplaced cells and final-pack the whole sequence;

14 Post-process self-symmetric cells to merge two split parts back to one unit;

End

Figure 4.4 Symmetric packing flow

The detailed procedure is illustrated in Figure 4.7 (a)-(d). Initially the two contour lists C_h and C_v only have boundary cell c_b and c_l respectively. And in the example, the dotted line shows the vertical contour and the dash line indicates the horizontal contour. After we place the first cell a, C_h and C_v are updated to $\{c_b, a\}$ and $\{c_l, a\}$, respectively. As the second cell d has horizontal relationship with cell a, it is positioned on the right of cell a, C_h is updated to $\{c_b, a, d\}$, and C_v is updated to $\{c_l, d\}$. For the next cell c that has horizontal relationship with cell d, it is placed on the right of cell d based on the current vertical contour list. Next, it is the turn of cell b_l to be placed, which is supposed to be on the top of cells a, d, and c. As currently C_h is $\{c_b, a, d, c\}$, we trace the cells in C_h so that cell b_l is placed according to the highest horizontal contour (i.e., cell d) as shown in Figure 4.7(d). After that, C_h is updated to $\{c_b, b_l, d, c\}$, C_v is updated to $\{c_l, c, d, b_l\}$. Up to now we have completed the packing of the cells on the left of symmetry axis DN1, whose coordinates can be determined accordingly.

In the following procedure, the coordinates of the symmetry axis need to be updated after placing each cell. When we process the first symmetric cell, the coordinates of the symmetric axis X_{axis} is just the right end of that cell. After that, if a placing cell belongs to a symmetric pair, X_{axis} is updated with the right end of that current cell if the latter is larger. If a placing cell is asymmetric, we will calculate the middle coordinates axis_{temp} between the right end of the current cell and the right end of the closest symmetric cells placed on the left. X_{axis} is updated with axis_{temp} if the

latter value is larger. In the example of Figure 4.7, we can see that the coordinates of the first symmetry axis DN1 are X_{axis1}. After we place the first symmetric cell *a*, $X_{axis1}=L_a$ where L is the width of the cell. After placing *d* that is an asymmetric cell, X_{axis1} is updated to $L_a + L_d / 2$. After we process cell *c* that is symmetric, X_{axis1} changes to $L_a+L_d+L_c$, which is the same as the horizontal boundary. And the second sub-sequence can be packed in the same way to obtain the horizontal coordinates of the axis.

Step 3: tune-up operation on symmetric cells with respect to the symmetry axis for meeting the symmetry constraints (Lines 8-10).

After initial packing, the vertical and horizontal positions of symmetric cells are modified with respect to the corresponding axis as listed in Figure 4.4. To avoid cyclic shift, we follow the packing sequence of *packSeq* to conduct the tune-up operation. After the comparison of Y coordinates of two cells within one symmetric pair (such as (b, b')), the lower cell would be shifted to the same level (i.e., $Y_b = Y_{b'} = max\ (Y_b, Y_{b'})$). Then we increase the vertical coordinates of its fan-out symmetric cells by ΔY. Notice that we need to shift the counterpart of the shifted cell as well.

For each two cells belonging to one symmetric pair (such as (b, b′)), $\Delta X_b = |X_b - X_{axis}|$ and $\Delta X_{b'} = |X_{b'} - X_{axis}|$ are calculated and one of the symmetric cells is shifted to make $\Delta X_b = \Delta X_{b'} = max\ (\Delta X_b, \Delta X_{b'})$. In addition, if the shifted cell is on the right (or left) with respect to the symmetry axis, its fan-out (or fan-in) symmetric cells in G_h are moved in the same direction with the same amount. This tune-up operation

guarantees that the symmetric cells are placed in harmony with the symmetry constraints. The examples can be seen from Figure 4.7 (f)-(g). From the graph, we can see that we shift cell b_r up with ΔY, and shift cell a' to the right with ΔX. Afterwards the third sequence (g', f') is packed and the tune-up operation is conducted for symmetry group 2 as shown in Figure 4.7(h).

Step 4: final packing of the entire TCG and post-processing of self-symmetric cells (Lines 13-14).

In the final packing, the same contour-based scheme is used for the entire TCG, in which the symmetric cells are considered as the preplaced ones (i.e., unchanged coordinates). That is to say, for a symmetric cell, it is just added to the contour list without modifying the coordinates because symmetric cells have been placed symmetrically with respect to the corresponding symmetry axis in the previous steps. Any shift of symmetric cells in the final packing would make all the efforts spent in the initial packing and tune-up operation in vain. In effect, the final packing is focused on positioning asymmetric cells compactly. As the contour lists are deployed, any waste space can be reused by asymmetric cells so that the entire placement area may be reduced.

In the previous steps, by dividing self-symmetric cells into two halves (i.e., the left and right ones), we can consider them as normal symmetric pairs. As special care has been taken in the perturbation operation (detailed in Section 4.3), it is guaranteed that there is no cell in between these two halves. As the last step of the packing, a

post-processing operation is conducted to shift the two halves towards the symmetry axis and eventually merge them to one cell. Since there is no cell placed between these two halves and they are considered to be symmetric pairs in the previous phases, this post-processing step is to locate the self-symmetric cells along the symmetry axis without fail.

Begin

1 Construct the topological order of the TCG;

2 Generate the packing sequence of *packSeq*, which contains only the symmetric cells;

3 Consider the dummy axis cells as barriers to separate *packSeq* into sub-sequences;

4 Create a *list* made up of the sub-sequences;

5 Do *initial packing* for the first sub-sequence and calculate the axis position X_{axis};

6 **For** (the *list* is not empty)

7 Do *initial packing* for the second sub-sequence;

8 Follow *packSeq* one by one. Compare the Y coordinates of each two cells in one symmetric pair such as (a, a'), and make $Y_a = Y_{a'} = max(Y_a, Y_{a'})$;

9 Follow *packSeq* one by one, for every two cells in one symmetric pair such as (a, a'); calculate $\Delta X_a = |X_a - X_{axis}|$ and $\Delta X_{a'} = |X_{a'} - X_{axis}|$. Then shift one symmetric cell to make $\Delta X_a = \Delta X_{a'} = max\ (\Delta X_a, \Delta X_{a'})$.;

10 Do final packing for the first and second sub-sequences;

11 Remove the first sub-sequence from the sub-sequence *list*;

12 **End for**

13 Post-process self-symmetric cells to merge two split parts back to one unit;

End

Figure 4.5 Simplified Symmetric packing flow

Assume we need to place n cells where p symmetry groups are included. Each

group has at most g symmetric pairs and s self-symmetric cells. And each sub-sequence has at most m cells. Our proposed packing scheme above first takes $O(n)$ time to generate the topological order (due to [28]). According to [16], the time complexity of the contour-based packing algorithm for each sequence is $O(m \cdot lgm)$. And we need $O(n)$ time to update the symmetry axes. For the tune-up operation of symmetric cells, we need $O(p \cdot (g + 2s))$ time for both the Y and X dimensions. In addition, the final packing takes $O(n \cdot lgn)$ time and the post-processing operation takes $O(p \cdot s)$ time. Therefore, for the worst case, the time complexity is $O(p \cdot n \cdot lgn)$ in total for our proposed packing scheme of TCG.

Based on the general packing scheme above, we can allow for a relatively simple implementation where the cells of different symmetry groups do not interfere with one another by placing all the symmetric cells of one group to the left (or right) of the symmetric cells belonging to another group as shown in Figure 4.1(I). For this situation, the algorithm is simplified by keeping Lines 1-7. And for Lines 8-9, we only need to tune up the coordinates of the symmetric cells since the dummy node representing the axis is used as a barrier, and the cells from different groups cannot interfere with one another. The simplified symmetric packing flow is presented in Figure 4.5. For this simplified scheme, for every loop, the tune-up operation takes $O(j)$ time. The loop number of the initial and final packing operations is of $O(p)$. And the post-processing step takes $O(p \cdot k)$ time. So the total complexity of this simplified packing scheme is $O(p \cdot m \cdot lgm)$. The experimental and comparison results between the general and simplified packing schemes will be detailed in Section 4.4.

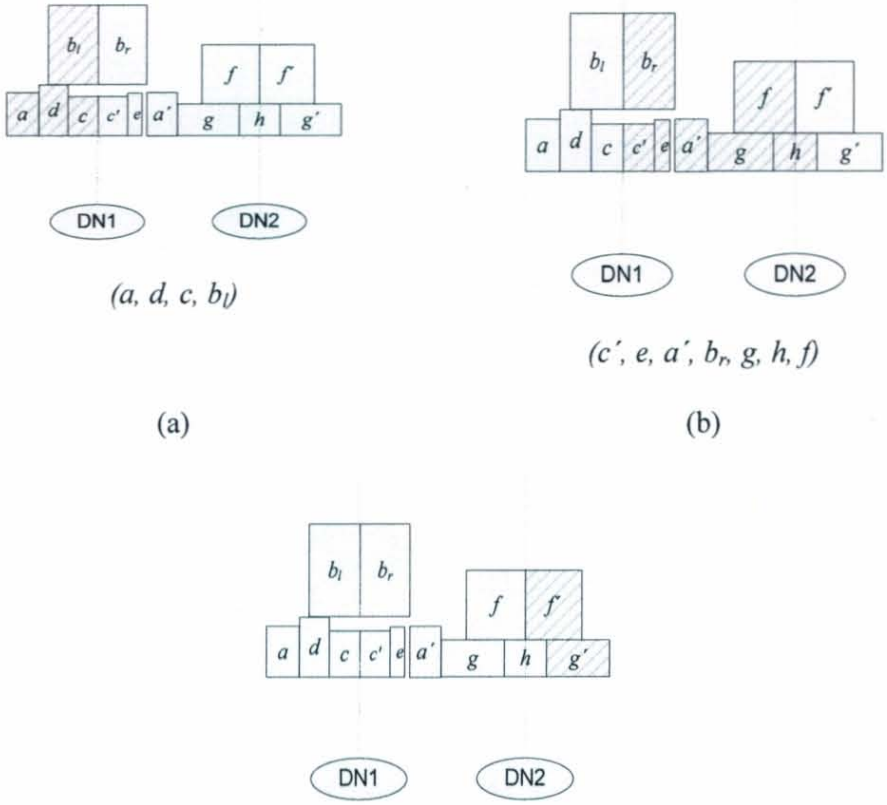


Figure 4.6 Packing sub-sequences

Compared to the HB*-tree approach [9], this proposed method can handle the situation where an asymmetric cell is placed between two symmetric cells (such as cell *e* in Figure 4.1). In contrast, following the symmetry-island definition in [9], symmetric cells in one symmetry group must be always connected. As a matter of fact, if an asymmetric cell includes some highly sensitive nets that are associated with both

cells in a symmetric pair, placing the cell within the symmetry group can significantly reduce the cost of wire length.

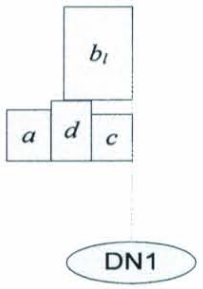
As a summary of the operation above, we can derive the following lemma:

Lemma 2: Any symmetric-feasible TCG containing multiple symmetry groups can be packed to a symmetric placement within polynomial time.

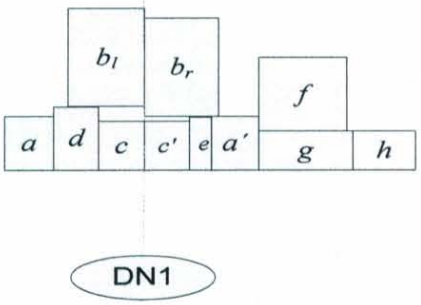
Proof: Following either packing scheme explained above, any symmetric-pair cells are placed at the same Y coordinates and at the same distance (on the opposite sides) from the corresponding axis since the coordinates of the symmetry axis are calculated based on this principle embedded within the tune-up operation. The self-symmetric cells are placed along the symmetry axis by the post-processing operation. Furthermore, as we use the contour-based packing scheme, the asymmetric cells would not overlap. And the complexity analysis shows the entire algorithm takes $O(p \cdot n \cdot lgn)$ time at most. Thus, this lemma is proved. ∎

(a) (b)

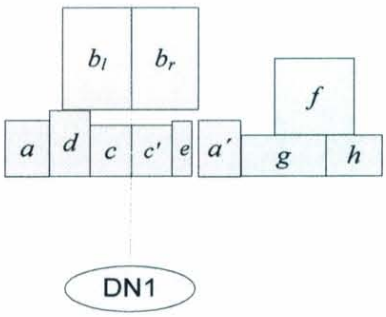
(c) (d)



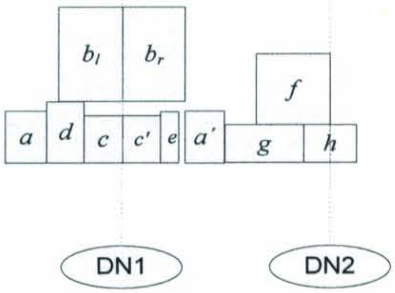
(e)



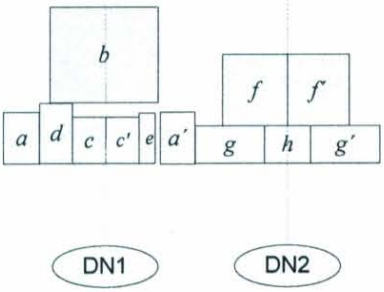
(f)



(g)



(h)



(i)

Figure 4.7 Example of the packing

Theorem 1: Optimal symmetric placement can be derived by searching the configuration space by means of symmetric-feasible TCG.

Proof: From Lemma 1, we have proved that for a symmetric placement, we can always find a symmetric-feasible TCG to represent it. And from Lemma 2, we are able to pack a symmetric-feasible TCG containing multiple symmetry groups to a symmetric placement. Thus, it is seen that symmetric-feasible TCGs and symmetric placements have a straightforward mapping relationship. By taking advantage of the correlation between both, a thorough search in the configuration space of symmetric-feasible TCGs can eventually pinpoint to a symmetric-feasible TCG state that maps the optimal symmetric placement. ∎

4.3 Symmetric-Feasible TCG Perturbations

To search for an optimal solution to symmetric placements, we need to conduct continuous random perturbation of TCG states, where the TCG validity and symmetric-feasibility should be always preserved. For this purpose, we introduce five perturbation operations in this section. Since all the perturbations are done by TCG edge operations, we will first prove that the topology-relationship change among the vertices in a TCG can be done in $O(n)$ time. In contrast, the same task has to be fulfilled in $O(n^2)$ time by means of checking the reduction edge in the conventional

TCG method [11].

4.3.1 Cluster Edge Move-Reverse and Edge Move Operations

Following the terminology defined in [11], in a TCG, moving an edge from G_h to G_v, or vice versa, is called *edge move* operation; moving an edge from G_h to G_v, or vice versa, and also changing the direction of the edge after the edge move is called *edge move-reverse* operation. We further have the following definitions:

Definition 2: The sum of in-degrees in both G_h and G_v of a vertex is called *in-in degree*, whereas the sum of in-degree in G_h and out-degree in G_v of a vertex is called *in-out degree*. For any two vertices (termed as reference vertices) in a TCG, an aggregate of the vertices, whose in-in degrees (or in-out degrees) are between the in-in degrees (or in-out degrees) of the reference vertices, is called *in-in* (or *in-out*) *cluster*.

Definition 3: *Cluster edge move-reverse* (or *cluster edge move*) operation is defined as a set of edge move-reverse (or edge move) operations between one reference vertex and each vertex from a union of the other reference vertex and in-in (or in-out) cluster vertices.

Lemma 3: Without losing TCG transitive-closure property, the topological relationship between any two vertices in a TCG can be modified by a cluster edge move or cluster edge move-reverse operation, which takes *O(n)* time.

Proof: As mentioned in Section II, TCG and SP are equivalent in functionality and different only in certain aspects. According to [28], for a TCG, the vertices in the α-sequence of the corresponding SP are ordered incrementally according to the in-out degrees of vertices, whereas the vertices in the β-sequence are ordered incrementally according to the in-in degrees of vertices.

As per Definition 2, the vertices within an in-in cluster are actually the ones between two reference vertices in the β-sequence. Thus, one cluster edge move-reverse operation is equivalent to reversing one reference vertex with respect to the union of the other reference vertex and in-in cluster vertices. Thus, the resultant TCG must remain transitive-closure as the corresponding reversed SP is valid. A similar observation can be conducted for a cluster edge move operation, where the only difference for this situation is that the vertices within an in-out cluster are actually the ones between two reference vertices in the α-sequence.

For a TCG, it takes constant time to obtain in-degree and out-degree of a vertex. And any trivial edge move-reverse or edge move operation only needs constant time. Therefore, one cluster edge move-reverse or edge move operation would take at most *O(n)* time. ∎

As an illustration shown in Figure 4.8(a), we follow the example used in [11] which has 5 vertices.

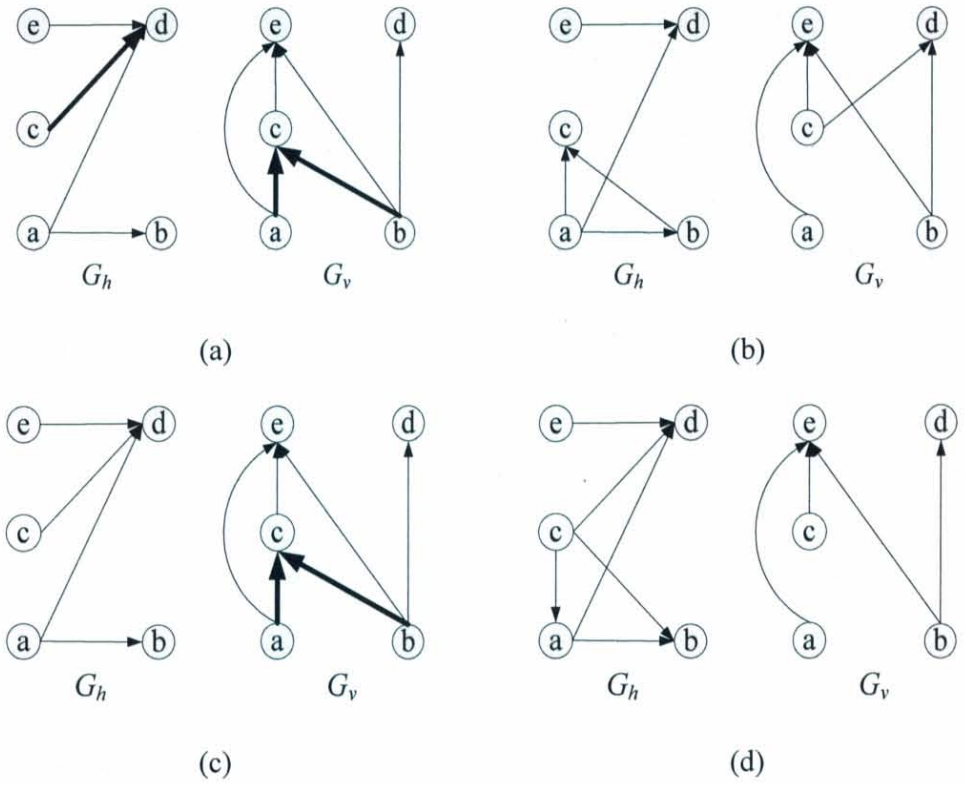


Figure 4.8 Perturbation example

First we run a cluster edge move operation to change the relationship from $b \perp c$ to $b \vdash c$. By calculating in-out degrees, we can determine the in-out cluster between vertices b and c is $\{a, d\}$.Thus, besides an edge move operation between vertices b and c, we have to make edge move operations between vertex c and any vertex from $\{a, d\}$ as shown in Figure 4.8(a). And the result after cluster edge move operation is

shown in Figure 4.8(b). It is obvious that the TCG transitive-closure feature is preserved but with no need to check the reduction edge [11]. Similarly, if we run a cluster edge move-reverse operation to change the relationship from $a \perp c$ to $c \vdash a$, the in-in cluster between vertices a and c that we determined is $\{b\}$. Thus, besides an edge move-reverse operation between vertices a and c, we have to make another edge move-reverse operation between vertices b and c which are bold as shown in Figure 4.8(c) and the result is shown in Figure 4.8 (d).

4.3.2. Five Perturbation Operations

In the following sub-sections, we will discuss five proposed perturbations. We declare that we use the dummy axis vertices to separate the TCG into different *regions*. From left to right, the first region is all fan-in vertices of the first axis vertex. And each of the next regions consists of in-in cluster between two adjacent axes. The last region includes all the fan-out vertices of the last axis vertex.

1) Vertex Rotation

Vertex rotation operation is actually to change orientation of the corresponding vertex. For a symmetric pair, rotation of one vertex should make the corresponding counterpart on the other side change to the mirror orientation with respect to the

symmetry axis.

Lemma 4: Given a symmetric-feasible TCG, the perturbed TCG is still symmetric-feasible and valid under the vertex rotation operation, and this operation takes *O(1)* time.

Proof: After the vertex rotation process, the vertices and edges of the TCG will remain the same as before. As we only need to exchange the weights of the related edges, the resultant graphs are still symmetric-feasible and valid. And exchanging the weights of the related vertices in G_h and G_v only takes *O(1)* time. ∎

2) Symmetric Swap

Symmetric swap operation is defined as follows: one vertex in a symmetric pair swaps position with its symmetric counterpart.

Lemma 5: Given a symmetric-feasible TCG, the perturbed TCG is still symmetric-feasible and valid under the operation of symmetric swap, and this operation takes *O(1)* time.

Proof: When processing the symmetric swap operation, we only need to exchange the two vertices and the corresponding edges. So the topology of the TCG, which was

originally symmetric-feasible and valid, does not change. The change of the two related vertices and the corresponding edges in the G_h and G_v of TCG only take *O(1)* time. ∎

3) Symmetric-Cell Move

This operation is to change the horizontal or vertical relationship between symmetric vertices within one symmetry group or from different symmetry groups. For example, if one symmetric pair was located at the bottom of one self-symmetric cell, the self-symmetric cell may be moved to a position within or at the bottom of the symmetric pair.

As mentioned in Section 4.2, the self-symmetric cells in our TCG are divided into two halves. To deal with this perturbation, we need to ensure there is no vertex to be placed within the halves. When the topology relationship of one cell from a symmetric pair changes, the relationship of its counterpart has to be changed accordingly if there is a violation of (4.1)-(4.4).

The details of the algorithm are shown in Figure 4.9. Recall that we use the dummy axis vertices as barriers to separate the TCG to different regions. As shown in Figure 4.10, the two dummy axis vertices divide the TCG into three regions. Following this operation, we first randomly pick up two symmetric vertices in one region (say, vertices *a* and *b)*, and randomly change their topology relationship. To keep the symmetric-feasibility, if *b* is a self-symmetric half, we cannot change to the

situation where vertex a is placed between the dummy axis vertex and vertex b. According to Lemma 3, we can safely change the vertices relationship based on in-in or in-out cluster. After we have made the relationship change, the counterpart vertices of the changed symmetric ones have to be verified. If there is a violation of (4.1)-(4.4), the counterpart vertices have to be updated according to the moved part. Note that the change of the counterpart vertices would not interfere with the original vertices as they are belonging to the different regions separated by symmetrix axes.

One example is shown in Figure 4.10. Assume vertices a and c from symmetric group 1 are randomly chosen. Due to $a \vdash c$, the derived in-out cluster for vertices a and c is $\{d\}$. After cluster edge move operation, the following two edges appear: $a \perp c$ and $d \perp c$. In addition, we need to change the relationship between the symmetric counterparts, that is, $a' \perp c'$. Following the same strategy, we can change the relationship from $c \perp b_1$ to $c \vdash b_1$. The result of the perturbation is shown in Figure 4.10(e), where the moving vertices are shown in the TCG.

Begin

1 Randomly pick up two symmetric vertices a and b in reference to a symmetry axis;

2 Check the relationship of these two symmetric vertices;

3 **IF** (a is one self-symmetric half vertex)

4 In reference to the dummy node (marked DN) for the symmetry axis, randomly change (by using the cluster-based edge operations) to another relationship between a and b except for $a \vdash b \vdash$DN or DN $\vdash b \vdash a$;

5 **ELSE**

6 Randomly change the relationship of a and b (by using cluster-based edge operations);

7 **ENDIF**

8 **IF** (there is a violation of (4.1)-(4.4) for the symmetric counterparts of vertices a and b)

9 Change the counterparts to the same relationship (by using cluster-based edge operations);

10 **ENDIF**

End

Figure 4.9 Symmetric cell move

Lemma 6: Given a symmetric-feasible TCG, under the operation of moving symmetric vertices, the perturbed TCG is still symmetric-feasible and valid. And this operation takes $O(n)$ time.

Proof: Following the scheme in Figure 4.9, the topology-relationship change between

symmetric vertices is always in accordance with (4.1)-(4.4). Thus, symmetric-feasibility can always be preserved. In addition, as the aforementioned geometry-relationship change is based on cluster edge move-reverse and cluster edge move operations, the updated TCG is still valid according to Lemma 4. As at most two cluster edge move-reverse or cluster edge move operations are involved, the complexity of this symmetric-cell move operation is just $O(n)$. ∎

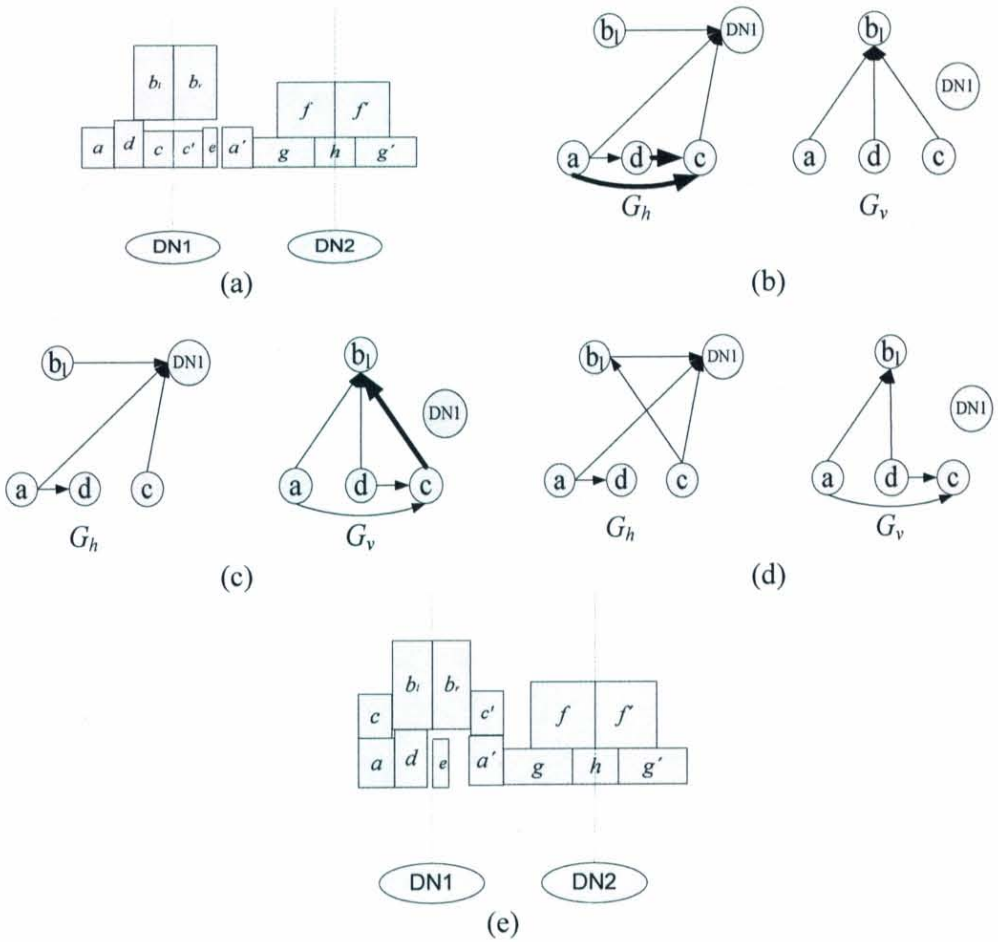


Figure 4.10 Example of symmetric-cell move

4) Asymmetric-Cell Move

This operation is to perturb the topology relationship between an asymmetric vertex and other vertices. The operation flow is listed in Figure 4.11.

Begin

1 Randomly pick up an asymmetric vertex *c* and another vertex *d* in one region;

2 **IF** (*d* is an asymmetric vertex)

3 Randomly change the relationship between *c* and *d* using the cluster-based edge operation;

4 **ELSE IF** (*d* is a self-symmetric vertex)

5 In reference to the dummy node (marked DN) for the symmetry axis, randomly change the relationship between *c* and *d* except for *d* ⊢*c* ⊢DN or DN ⊢*c* ⊢*d*;

6 **ELSE IF** (*d* is one dummy node for symmetry axis)

7 Pick up *c* and *d* as reference, and all the vertices in the target regions is in the in-in cluster;

8 Apply the cluster edge move-reverse operation and edge move operation to update the TCG;

End

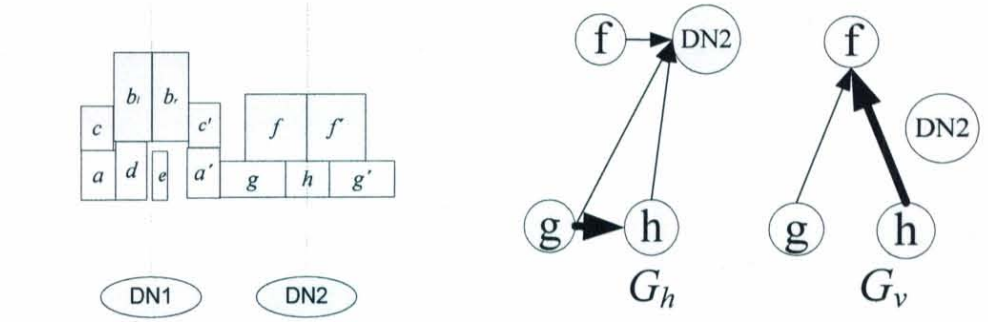
Figure 4.11 Asymmetric cell move

First we randomly pick up an asymmetric vertex c and another vertex d in one region. If d is an asymmetric vertex, we randomly change the topology relationship between c and d. If d is a self-symmetric vertex, we randomly change the topology relationship between c and d but excluding the situation where c is eventually placed between d and its corresponding dummy symmetry axis vertex. If d is a dummy symmetry axis vertex, since any vertex can only have horizontal relationship with the dummy symmetry axis vertex, this topology-relationship change is to place the asymmetric cell (i.e., c) to another region. So when this operation is applied, we take vertex c and the dummy node vertex as two reference vertices. All vertices C_i in this region are in the in-in cluster of the two vertices. And then we apply the cluster edge move-reverse operation and edge move operation to update the TCG.

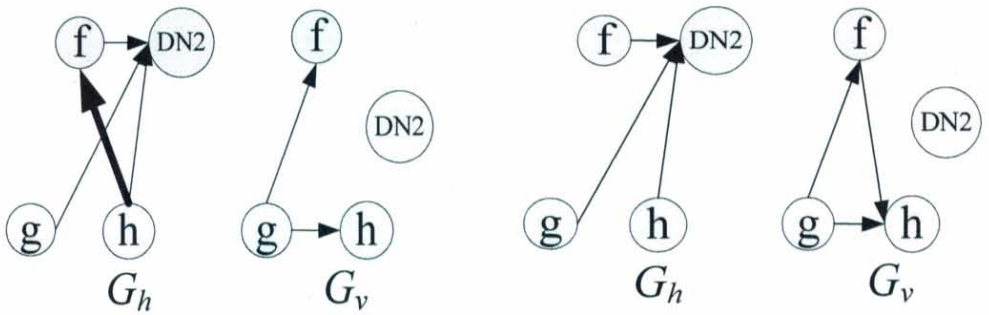
As an illustration, Figure 4.12 exhibits the topology-relationship changes between asymmetric cell h and symmetric cell f. In the graphs, we only present the related vertices. The in-out cluster between h and f is $\{g\}$. To update the graphs, we can use cluster edge move-reverse to change the relationship of h with reference to vertex g and f. And then we can apply the cluster edge move and reverse operation between vertex f and h. Figure 4.12(d) depicts the new placement after the perturbation where h is moved to the top of symmetric cell f.

Lemma 7: Given a symmetric-feasible TCG, under the operation of moving asymmetric cells described above, the perturbed TCG is still symmetric-feasible and valid. And this operation takes $O(n)$ time.

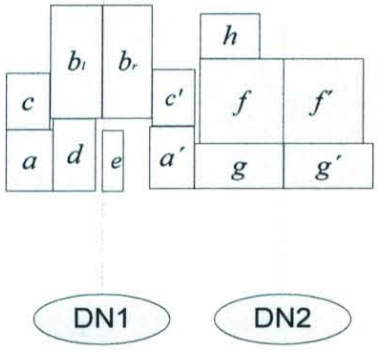
Proof: Following the process in Figure 4.11, the topology-relationship change of asymmetric vertices is always in accordance with (4.1)-(4.4). Thus, symmetric-feasibility can always be preserved. In addition, it first takes *O(1)* time to pick up the vertex, and then *O(n)* time to extract the in-out or in-in cluster. According to Lemma 3, the eventual cluster edge move-reverse or cluster edge move operations only needs *O(n)* time to construct a new symmetric-feasible and valid TCG. ∎



(a) (b)



(c) (d)



(e)

Figure 4.12 Example of asymmetric move

5) Symmetry-Group Move

This operation is to change the relative position of different symmetry groups. The flow is detailed in the Figure 4.13. We can randomly pick up one dummy axis vertex for perturbation, and we are going to place the related symmetry group at the most right position. To apply this operation, we first delete all vertices belonging to this symmetry group (with respect to the selected dummy axis vertex). The TCG is still transitive closure since we also delete all edges related to the deleted vertices. And then we begin to add the deleted vertices follow the in-in degree order one by one. We set them to be the fan-out vertices to all vertices currently in G_h and no relationship in G_v. One example of the symmetry group move operation is shown in Figure 4.14. In other words, we remove all the cells c_i belonging to symmetry group 1,

and then place all these cells to the rightmost side of all the remaining cells. As they are placed to form a group, by further using operations 1) - 4) described above, we can continue to search for other configurations.

Begin

1. Randomly pick up one symmetry group A, search all cells in this group.
2. Delete all vertices belonging to this symmetry group A with respect to the picked dummy axis vertex;
3. Set them to be the fan-out vertices to all vertices current in G_h and no relationship in G_v;
4. Output the new TCG

End

Figure 4.13 Asymmetric cell move

Lemma 8: Given a symmetric-feasible TCG, under the operation of symmetry group move, the perturbed TCG is still symmetric-feasible and valid. And this operation takes *O(n)* time.

Proof: It first takes *O(1)* time to pick up the related vertices and then *O(n)* time to delete the vertices. Then it takes *O(n)* time to complete the insertion process. Moreover, as the newly formed symmetry group is placed in the in-in degree order and only has horizontal relationship among the related vertices, the TCG is still symmetric-feasible. ∎

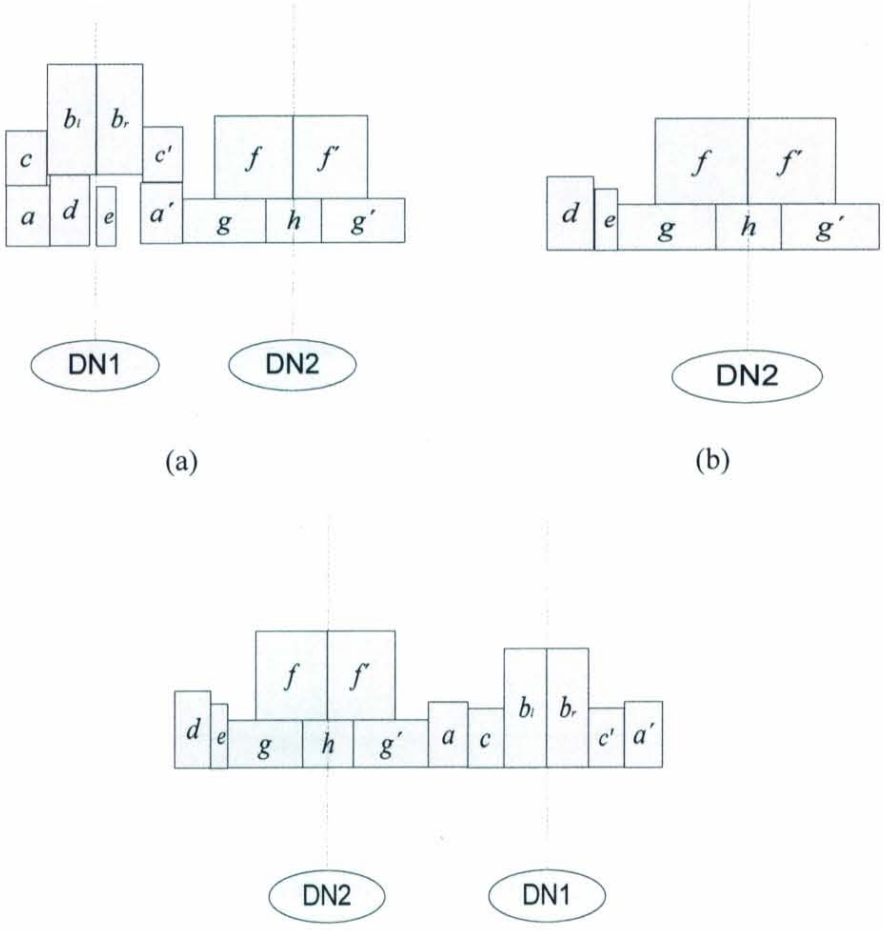


Figure 4.14 Example of symmetric group move

Theorem 2: By means of operations 1) - 5) detailed above, the solution space of the symmetric-feasible TCGs can be fully explored. Each operation takes at almost *O(n)* time, where *n* is the number of cells.

Proof: Since all the perturbations in 1) - 5) are done by TCG cluster edge move-reverse or cluster edge move operations, they can be completed in *O(n)* time. And in our perturbation scheme, we are able to randomly change the relationship of any two vertices unless there is violation to the symmetric-feasible conditions (4.1)-(4.4). Therefore, the continual perturbations and search using operations 1)-5) can ensure to traverse the complete configuration space of placements. ∎

4.4 Experimental Results

In this section, the experimental results using our proposed scheme are reported. They are compared with several other methods to evaluate the performance of our method. The final placement of several test circuits by using our scheme is also shown in this section.

Following the algorithm described above, I have implemented the proposed symmetry-aware TCG scheme based on a simulated annealing optimization flow. The program was coded in C++ language under Linux operating system and tested on a 2GHz PC with distinct test benchmark circuits. Compared to the existing schemes, our method is able to effectively search the entire configuration space. And it theoretically features less perturbation complexity than TCG [11] or TCG-S [16], and less packing complexity than SymmTCG [28]. In addition, this method can handle

multiple symmetry-group constraints. The cost function is constructed in the following format:

$$Cost = \alpha_{area} \cdot Area + \sum_i \beta_i \cdot WireLength_i , \qquad (4.5)$$

where α and β are two factors prioritizing the weights between area and wire length. And for different wires or nets, different β_i can be used. As we pay more attention to improving the placement quality of analog layouts, the cost values obtained from distinct approaches are of utmost concern, and we are less worried about the computation time compared to the cost.

Since there are only a few published approaches available for the multiple symmetry-group placement, we compare our work (i.e., S-TCG-1 representing the complete flow with the general packing scheme and S-TCG-2 standing for the flow with the simplified packing scheme discussed in Section 4.2) with the absolute placement method [10], basic SP [23], SP with dummy node [24], SP with linear programming [26] and HB*-tree method [9]. In this work, we have used two groups of test circuits. In the first group, three MCNC benchmark circuits (i.e., apte, ami33, and ami49) were modified to include two symmetry groups in each circuit. The second group includes three industry-size analog circuits, each of which is composed of 60-100 cells and 3-5 symmetry groups. In addition we use an OTA from [41] as our own test circuit. The details of the test circuits and the compared approaches are listed in Tables 4.1 and 4.2.

First we test the performance of the distinct placement algorithms on the MCNC

benchmark circuits and the results are shown in Table 4.3. We list the number of the cells and the number of symmetry groups in each circuit. The last column shows the absolute cost value and execution time of our *S-TCG-2* method. The cost value is calculated from the cost function including both area and wire length. And in the cost function, we have assigned distinct factors to different nets. The other columns show the percentage of the result value of the other methods we implemented compared to our proposed *S-TCG-2* method.

Table 4.1 MCNC benchmarks

index	Name	Block size	Nets	Symmetric groups	remarks	Source
1	apte	9	97	2		MCNC
2	ami33	33	123	2		MCNC
3	ami49	49	408	2		MCNC
4	biasynth _2p4g	65	100	3	Modified	Modified from [9]
5	lnamixbias_ 2p4g	110	100	5	Modified	Modified from [9]
6	Mod_ biasynth	65	100	3	Modified	Modified from [9]
7	Mod_lnamixbias	110	100	5	Modified	Modified from [9]
8	OTA	69	100	5	Modified	[41]

Table 4.2 Approaches for comparison

Index	Name	Technique	Packing	Perturbation	Ref.
1	Abs	Absolute mothod	$O(n^2)$	$O(n^2)$	[10]
2	SP	SP	$O(n^2)$	$O(1)$	[23]
3	SPWD	SP+ dummy node	$O(n^2)$	$O(1)$	[24]
4	SP+LP	SP+Linear Programming	$O(n^2)$	$O(1)$	[26]
5	HB*-tree	Hierachical B*-tree	$O(n)$	$O(lgn)$	[9]
6	S-TCG-1	TCG	$O(p \cdot nlgn)$	$O(n)$	
7	S-TCG-2	TCG	$O(p \cdot mlgm)$	$O(n)$	

Table 4.3 Results of MCNC benchmarks

Circuit		***Abs***	***SP***	***SPWD***	***SP+LP***	***HB*-tree***	***S-TCG-1***	***S-TCG-2***
apte	Cost	120.7%	112.2%	107.7%	106.4%	101.2%	100.0%	50.74
	Time (sec)	1100%	767%	466.7%	400.0%	100%	119%	3
ami33	Cost	118.9%	109.3%	108.5%	107.4%	103.7%	99.0%	1.33
	Time (sec)	1670%	1190%	244.4%	354.8%	104%	121%	62
ami49	Cost	129.1%	107.9%	108.7%	107.1%	102.5%	99.2%	42.21
	Time (sec)	2250%	1360%	201.0%	228.0%	113%	118%	107
average	Cost	124%	110%	108%	107%	102.5%	99.4%	
	Time (sec)	1640%	1105%	274%	327%	105.6%	119%	

From the experimental results listed in Table 4.3, we can find that our methods achieve the best performance in terms of cost. On average, *S-TCG-2* can reduce the cost by 24% compared to *Abs*, 10% compared to *SP*, 8% compare to *SPWD*, 7% compared to *SP+LP* and 2.5% compared to *HB*-tree*,. In terms of execution time, we can see that *S-TCG-2* is 15.4 times faster than *Abs*, 10 times faster than *SP*, 1.7 times faster than *SPWD*, 2.3 times faster than *SP+LP*, 5.6 % slower than *HB*-tree*. As suggested in [9], the tree representations such as *B*-tree* and *O-tree* intrinsically have low complexity. Therefore, *HB*-tree* can manage the packing in the linear time. However, this approach is based on the concept of symmetry-island and can only handle the situation where all symmetric cells are located together. No asymmetric cells or symmetric cells from another symmetry group can separate the symmetric cells of one symmetry group. This limitation would inevitably compromise the search quality for the final optimal placement.

Comparing two packing schemes discussed in Section 4.2, we can see that the general packing scheme (i.e., S-TCG-2) is helpful (by decreasing the cost by 0.6% compared to the simplified packing scheme (i.e., S-TCG-1)), but at the expense of 19% more in terms of execution time. So in the following experiments we consider the simplified packing scheme as our standard implementation for the comparison with other approaches on the test of large-size analog circuits. Figure 4.15 shows the final placement result of ami33 with two symmetry groups. The symmetric cells are

shadowed for easy identification. The left symmetry group includes two symmetric pairs and one self-symmetric cell, whereas the right symmetry group has two symmetric pairs.

In addition, as shown in Table 4.4, we tested two analog circuits, including biasynth_2p4g of 65 cells and lnamixbias_2p4g of 110 cells [9] [24] [26]. Comparing with the other work, we have achieved 21% better than Abs, 12.2% better than SP, 9.4% better than SPWD and 7% better than SP+LP in cost on average. Also our method runs 21.9 times, 9.7 times, 4.5 times and 6.7 times faster than Abs, SP, SPWD and SP+LP, respectively. In addition, our proposed method is able to achieve certain improvement in terms of cost (2.1% and 4.2%, respectively) compared to the HB*-tree work, although the execution time of our method is inferior. From our observation, the symmetry groups in those test circuits are mainly made up of the cells with similar sizes and few self-symmetric cells. Thus, the best placement situation is to pack the symmetric cells together closely without any involvement of asymmetric cells in between, i.e., with the design methodology of symmetry island [10].

Table 4.4 Results of industry circuits

Circuit		*Abs*	*SP*	*SPWD*	*SP+LP*	*HB*-tree*	*S-TCG-2*
biasynth _2p4g	Cost	127%	109.8%	107.4%	104%	102.1%	5.42
	Time (sec)	2150%	611%	154%	221%	89.3%	131
lnamixbias_ 2p4g	Cost	119%	114.6%	109%	109%	104.2%	51.41
	Time (sec)	2430%	1130%	543%	920%	82.5%	287
average	Cost	123%	112.2%	108%	105.5%	102.7%	
	Time (sec)	2290%	871%	349%	667%	85.9%	

To evaluate the performance of distinct algorithms when handling more complex situations, we modified the two circuits by adding some self-symmetric cells to different symmetry groups and changing size of some symmetric-pair cells. In addition, we deployed another test circuit called OTA due to a highly linear operational transconductance amplifier [41], which includes 69 cells and five symmetry groups. The experimental results are listed in Table 4.5. On average, our proposed method reduced the cost by 21%, 11.2%, 9.4% and 7% compared to Abs, *SP, SPWD* and *SP+LP*, respectively. In particular, for mod-biasynth, mod-lnamix, and OTA, our method gained 4.2%, 5.1%, and 5.5% reduction in terms of cost compared to *HB*-tree*, respectively. As for the execution time, our method is faster

than *Abs*, *SP*, *SPWD* and *SP+LP* by 2121%, 718%, 128% and 373%, respectively. Among all the algorithms, *HB*-tree* tends to be the fastest method (around 11.6% faster than our method).

Table 4.5 Results of modified industry circuits and OTA

Circuit		***Abs***	***SP***	***SPWD***	***SP+LP***	***HB*-tree***	***S-TCG-2***
Mod_biasynth	Cost	123%	107.8%	107.1%	106%	104.2%	5.53
	Time (sec)	2137%	621%	153%	224%	88.5%	144
Mod_lnamixbias	Cost	120%	114.1%	111%	110%	105.1%	52.23
	Time (sec)	2440%	1110%	572%	934%	86.3%	294
OTA	Cost	119%	114.9%	107.2%	105%	105.5%	27.53
	Time (sec)	2084%	724%	167%	207%	90.4%	263
average	Cost	121%	112.2%	109.4%	107%	104.9%	
	Time (sec)	2221%	818.2%	288%	473%	88.4%	

Nevertheless, analog designers normally prefer to reply on design automation tools to conduct a thorough search at the expense of execution time. Therefore, in our optimization the effort on the reduction of cost should be primary, whereas the performance on the execution time is deemed as a secondary objective. Besides, as the execution time for a quite large analog circuit (say, 110 cells) is about two hundred seconds by using our proposed method, it would be acceptable from the

analog circuit designers' point of view if we consider the entire layout generation of a normal size analog circuit that may take a few hours. More importantly, compared with HB*-tree, our proposed scheme can handle more general placements since symmetric cells may not be always closely adjacent to each other.

One example placement of lnamixbias_2p4g with 5 symmetry groups is depicted in Figure 4.16. The grey area denotes the symmetric cells. We can see the area within the ellipse including three asymmetric cells placed between the symmetric pairs 70, 71, 72, and 74. However, the HB*-tree method is not able to handle this type of situations and cannot obtain such kind of placements. In practice, this feature offered by our proposed algorithm may reduce wire or area cost, and guarantee that our method can fully explore the placement solution space. The schematic of the OTA circuit is shown in Figure 4.17 and the final placement of OTA with 5 symmetry groups using our algorithm is shown in Figure 4.18.

In Table 4.6, we list the results from our implemented SP with linear programming method. Since [26] does not offer detailed perturbation operation, we employ the general SP perturbation for this work. We report the number of average times to find one successful SP and the number of successful SP in the first 10^7 runs. We can see that the *SP+LP* method spent a lot of time in finding a successful SP, which increased the whole execution time. But our method is able to find a valid symmetric-feasible TCG after each perturbation, which clearly saves time.

Table 4.6 SP with linear programming

Circuit	*Average times with one success*	*Success times with first 10000000 times run*
ami33	2278187	4
lnamixbias_ 2p4g	9201177	2

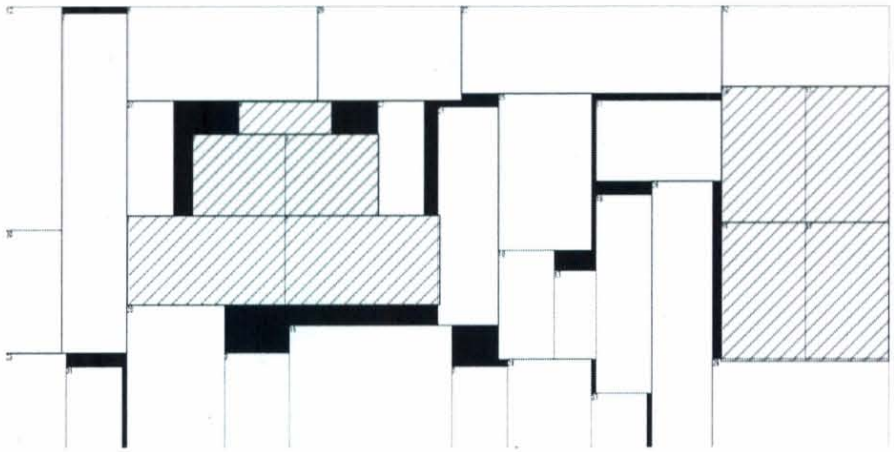


Figure 4.15 Final placement of circuit ami33

Figure 4.16 Final placement of circuit lnamixbias_2p4g

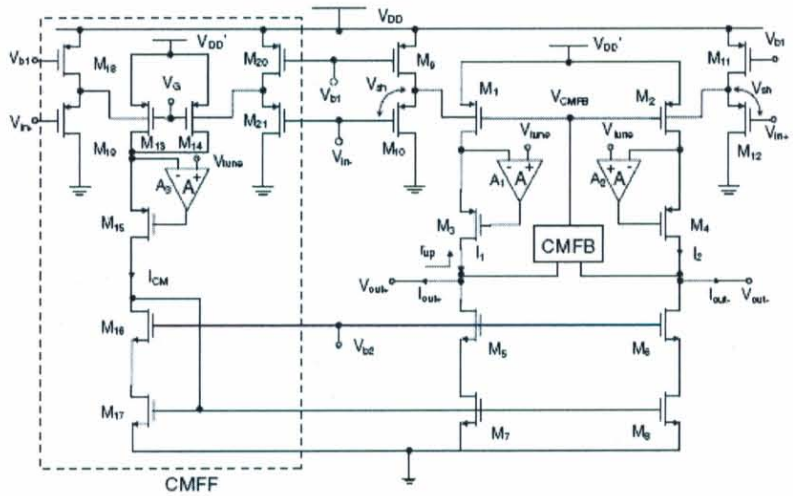


Figure 4.17 The schematic of the OTA

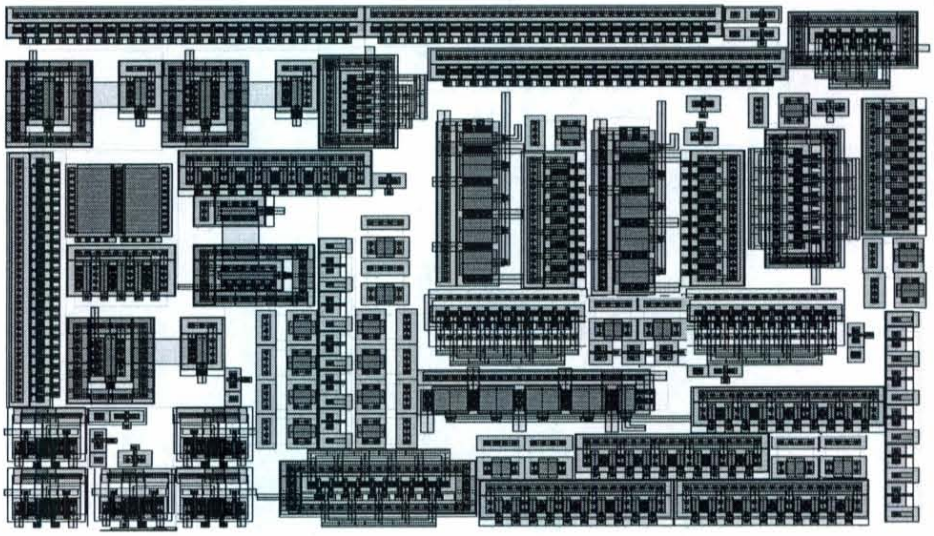


Figure 4.18 Final placement of circuit OTA

In summary, our method is efficient to handle the multiple symmetry group constraints. And the experimental results show that our method achieves far better performance compared to the *absolute method*, *SP*, *SPDW*, and *SP+LP* method. We are able to obtain even more impressive solutions compared to the HB*-tree method but with a slight trade off on the execution time.

4.5 Summary

In this chapter, I presented a complete set of symmetric-feasible conditions and a new packing scheme to handle the constraints of multiple symmetry groups for analog layout placement. An efficient perturbation strategy was proposed to achieve random

state conversion in *O(n)* time without losing TCG symmetric-feasibility and validity. Then I tested the proposed method based on a simulated annealing optimization flow with MCNC benchmark and several circuits. Our algorithm has shown better performance compared to several well-known methods.

Chapter 5 Substrate-sharing and Other Constraints

Besides the symmetry constraints, the TCG-based method is capable of handling another type of important constraints – substrate-sharing constraints, which have not been fully considered in any prior work. In this chapter (Sections 5.1-5.3), I will detail the formulation of the substrate sharing constraints and discuss how to handle these constraints in our proposed method. The experimental results are reported in Section 5.4. In Section 5.5, the solution to handling other analog layout constraints, including relationship constraints, proximity constraints, and alignment constraints, are also discussed. Finally a brief summary is made in Section 5.6.

5.1 Introduction

S*ubstrate-sharing constraints* mean some devices are placed adjacently so that they can share a common substrate/well region. This operation is normally followed by certain device geometry merging optimization [44]. In the analog circuits, some devices can be located as a connected or adjacent placement so that the devices can share a common substrate/well region, which decreases the effect of substrate

coupling [44]. One example is shown in Figure 5.1. In Figure 5.1 (a) two transistors M1 and M2 have their own substrates, whereas in Figure 5.1 (b) the substrates are merged to form one single substrate around the devices. In addition, M1 and M2 are merged to share one diffusion area. Obviously, by merging the substrate of the devices, the total area and the wire length can be minimized.

(a) (b)

Figure 5.1 An example of applying the substrate sharing constraint

P. Drennan *et al.* [43] addressed one significant substrate sharing effect, which is called shallow trench isolation (STI) stress. The shallow trench isolation, also known as "Box Isolation Technique", is an integrated circuit feature that prevents electrical current leakage between adjacent semiconductor device components. STI is generally used on CMOS process technology nodes of 250 nanometers and smaller. Older CMOS technologies and non-MOS technologies commonly use isolation based on LOCal Oxidation of Silicon (LOCOS). Reference [43] compared the performance between merged layouts and non-merged layouts taking into account several other aspects, including current mirror ratio, mismatch, maximum drain source voltage, and oxide defined area. The drawn conclusion is that the merged layouts achieve better

performance in the listed four areas compared to the non-merged layouts. Therefore, applying the substrate sharing constraints is very helpful in the analog circuit design. The advantages of these constraints are listed as follows:

1) These constraints can reduce the area of the placement since several devices can be placed tightly by sharing the common substrate.

2) The substrate sharing can reduce the complexity of routing for the analog layout. Besides, it is obvious that the merged devices can reduce the total wire-length of the placement.

3) Applying the substrate sharing constraints can decrease the coupling effect of substrate, minimize parasitic capacitance [8] during placement and enhance the performance of the circuit.

Cohn *et al.* [8] introduced a placement tool named KOAN to merge MOS devices in order to explore desirable optimization. In [8], eight typical types of geometry sharing optimizations in analog VLSI layout are described as shown in Figure 5.2.

a) MOS diffusion merging is the case when two or more MOS devices share a common source or drain diffusion, which reduces parasitic capacitances by eliminating some routing parasitics and reduce the area and perimeter of diffusion to bulk junctions.

b) MOS well merging is the case when the well regions of two or more MOS devices are connected by overlapping placement. Well merging lowers parasitic capacitance by reducing the area and perimeter of well to substrate junctions.

c) MOS bulk contact merging is the case when two or more MOS devices share

common bulk, well or substrate contacts. The capacitance reduction of the bulk contact merging is important.

d) MOS gate abutment routing is the case when an electrical connection between two or more devices' gates is made by non-overlapping abutting placement of polysilicon gates. It reduces parasitic capacitance and layout area by eliminating the need for discrete contacts and routing.

e) MOS strapping abutment routing is the case an electrical connection between two or more devices' source or drain strapping. And the effect of this sharing optimization is the same as that of gate abutment routing.

f) BJT collector merging is the case when an electrical connection between two bipolar collectors is made by overlapping placement of the collector regions. The BJT collector merging reduces critical capacitance on the device's collector regions.

g) BJT guard-ring merging is the case when an electrical connection between two or more bipolar devices' diffusion guard rings is made by overlapping placement. It is effective in reducing layout area by eliminating the need for minimum diffusion spacing between adjacent BJTs.

h) Capacitor abutment routing is the case when an electrical connection between two or more non-precision capacitors is made by non-overlapping abutment of the capacitor contacts. It reduces parasitic capacitance and layout area by eliminating the need for discrete contacts and routing.

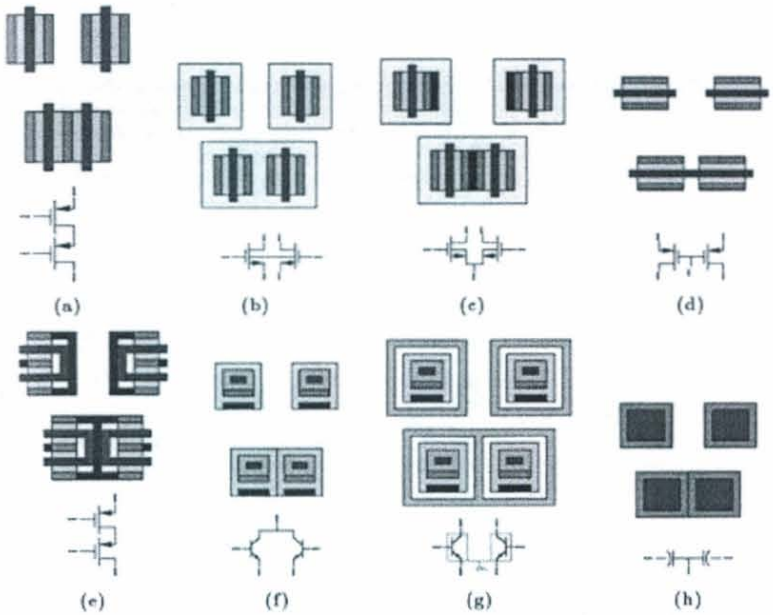
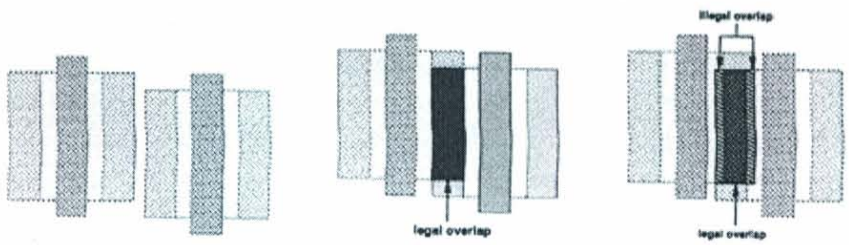


Figure 5.2 Various forms of device geometry sharing (a)MOS diffusion merge (b) MOS well merge (c) MOS bulk contact merge (d) MOS gate abutment (e) MOS strapping abutment (f) BJT collector merging (g) BJT guard-ring merging (h) capacitor abutment.

Focusing on the substrate sharing constraints, we may have three kinds of sharing situations in the layout as shown in Figure 5.3. In Figure 5.3 (a), the two devices are spaced far enough apart and have no illegal overlap. The second situation is shown in Figure 5.3 (b) where the two devices are placed by overlapping their substrate areas (called legal overlap). The last situation is an illegal overlap, which will be penalized in the KOAN method.



(a) (b) (c)

Figure 5.3 Use of protection frames in merge detection (a) no merging (b) complete merging, no illegal overlap (c) merging and illegal overlap.

Based on the situations mentioned above, KOAN [8] handles the substrate sharing constraints as a post-processing step following the placement. All the merge processes work as a post-processing step and have to place new restriction on the device generation, which generalizes the move-set to allow merged devices to move as a group. A new geometry-sharing encouragement term C_{merge} is added to the annealing cost function as follows.

$$C_{merge} = \sum_{i=1}^{N} \beta_i [\sum_{j=1}^{T_i} \varepsilon_{ij} (MergableArea_{ij} - MergedArea_{ij}) + \varsigma_{ij} (MergablePeri_{ij} - MergedPeri_{ij})] \quad (5.1)$$

$$C_{total} = \alpha_{overlap} C_{overlap} + \alpha_{length} C_{length} + \alpha_{area} C_{area} + \alpha_{separation} C_{separation} + \alpha_{orient} C_{orient} + \alpha_{aspect} C_{aspect} + \alpha_{merge} C_{merge} \quad (5.2)$$

The merge possibilities are not limited to a pre-defined set of module-generated structures. However, this method completely ignores the electrical and geometric implications. The shape of the well geometry cannot be determined before the placement of device is known. After the simulated annealing flow, the optimal merged placement solution is generated. Moreover, this method has extremely high time complexity due to the feature of the absolute placement scheme. In [42], a method named ALDAC can merge the MOS group. However, the paper failed to describe much about the details of how to merge the cells.

To the best of my knowledge, thus far there is no method based on topological representations for handling the substrate sharing constraints. I am motivated to propose the first topological-representation-based method to take care of the substrate sharing constraints in the placement design for analog layouts. Our experimental results show that this method achieves the goal to minimize the area and the wire-length of the circuits.

5.2 Substrate Sharing Problem Definition

To regard the substrate-sharing constraints, we define substrate area and orientation for each cell. The cells may have one, two or four substrates to share as shown in Figure 5.4. Each substrate has an orientation with respect to the orientation of the corresponding cells.

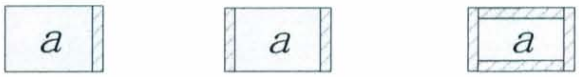


Figure 5.4 The substrate area of cells

If the cells are placed adjacent and the orientation of the substrate matches, we apply merge process to the cells as shown in Figure 5.5.

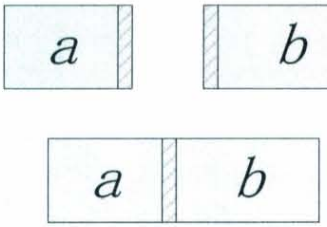


Figure 5.5 An example of merging

Also for the substrate sharing, we may have partial sharing situation as shown in Figure 5.6. This means we need to calculate how much area may be merged by two cells according to the adjacency situation of the cells and the substrate area of different cells. After we have the calculated results, we understand how much area we can merge for each cell.

Figure 5.6 Partial sharing and full sharing

Then we use the TCG representation to handle the processing since the TCG has an important feature, that is, the weight that is to indicate the distance between the corresponding two cells. In contrast, the SP representation cannot show the distance of two cells before packing. With this feature, the TCG representation can be used to readily handle the substrate sharing constraints.

5.3 Algorithm to Handle Substrate Sharing Constraints

To handle the substrate sharing constraints, we take advantage of the weight value to represent the merge situation as shown in Figure 5.7. For example, in the circuit, we have cells a and b, whose original weight of the edge from a to b in G_h is 10. And the substrate width is 2 for a and 2 for b. If the two cells are merged horizontally, we can update the edge weight to 10-2=8. That is to say, the edge weight of two cells would be less if the cells are sharing their substrate areas. Then we merge the cells with the substrate sharing area of the two cells.

Figure 5.7 Example of merge

Similar to handling the symmetry constraints, we modify the packing and perturbation schemes to deal with the substrate sharing constraints. The packing flow is listed in Figure 5.8. First, we generate the topological order of the cells. And then we use the *fan-in* (*fan-out*) degree to evaluate whether the cells are placed adjacently. Following the topological order one by one, we check the orientation of the substrate of the adjacent cells. If the orientation matches, these cells are able to share the substrate and can be merged. The area to be merged should be determined depending on substrate size and adjacent situation. The weight of the edges in the TCG of the merged cells will be updated accordingly.

```
Begin
  1   generate the topological order;
  2   FOR (the cells in the topological order one by one);
  3     Apply contour packing;
  4     IF (the substrates of the current cell and previous cell are placed
        adjacently and the orientation of the substrate matches)
  5       Apply the merge process;
  6       Update the corresponding contour;
  7     ENDIF
  8   END FOR
End
```

Figure 5.8 Substrate sharing packing flow

For the packing process, we follow the contour-based packing flow described in chapter 4 to process the cells. If the adjacent cells are merged, we will merge the substrate area of the cells and update the contour list. One example is shown in Figure 5.9. We employ the same example used in chapter 4. First, we place cell *a* that has one substrate as shown in Figure 5.9 (a). Then we place cell *b* with four substrates. Since the substrate orientation matches, and we can merge the substrate areas of cells *a* and *d* as shown in Figure 5.9(c). Then we need to update the contour list after the merging for further packing. The top contour of cell *a* is shortened by the substrate width. Then we place cell *c* with one substrate, which can be merged with cell *b* if their substrate orientations match. The merging of cells *b* and *c* is shown in Figure 5.9 (e). In addition, we need to update the top contour of cells *c* after the merging process.

Then cell b_l has been placed. Since cell b_l is a self-symmetric cell, we first place it to the correct location that is adjacent to the symmetric axis, and then merge the substrate depending on the adjacent situation as shown in Figure 5.9 (f). The final placement of (a,d,c, b_l) is shown in Figure 5.9 (h).

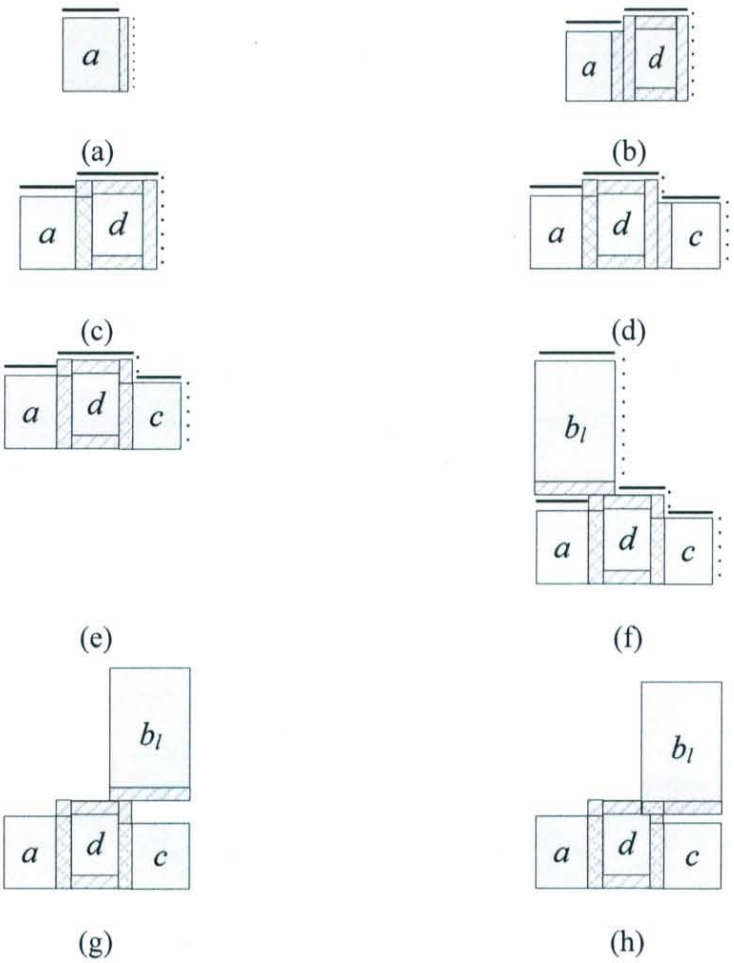


Figure 5.9 Example of the packing flow

The perturbation is the same as the method handling symmetry constraints. We only need to apply the perturbations of cell orientation change and asymmetric cell

move if we do not include the symmetry constraints. In this situation, we only consider all the cells are asymmetric cells and check the substrate sharing situation.

As mentioned in Chapter 4, the extraction of the topological order takes *O(n)* time where *n* is the number of the cells needed to be placed and the contour-based packing takes *O(nlgn)* time. The merge process takes *O(n) time*. In total, we have a packing scheme with the time complexity of *O(nlgn)*. For the perturbation, since it only takes time in the cell orientation change and asymmetric cell move, its time complexity is *O(n)* where *n* is the number of the cells.

5.4 Experimental Results of the Substrate Sharing Method

Based on the algorithm discussed above, I have implemented the proposed substrate sharing TCG scheme based on a simulated annealing optimization flow. The program was coded in C++ language under Linux operating system and tested on a 2GHz PC with distinct benchmark test circuits. The cost function is constructed as follows:

$$Cost=\alpha_{area}\cdot Area+\sum_i \beta_i\, WireLength_i, \qquad (5.3)$$

where α and β_i are two factors prioritizing the weights between area and wire length. Different nets may have different β_i due to distinct sensitivity to circuit performance. Since there is no previous work targeting at the substrate-sharing constraints by using the topological representations, we only compare the proposed substrate-sharing scheme with our symmetry constraint method (denote as S-TCG). Our substrate-sharing method is denoted as (S-TCG (m)). We can see the results shown in Table 5.1. One can see that the

substrate-sharing method can decrease the cost of the benchmarks with 4.8% on average. As for the large industry circuits, our substrate-sharing method can reduce the cost by 5.6% on average. Note that the running-time performance is not our focus at the moment. One example of the final placement of APTE is shown in Figure 5.10 and the final placement of lnamixbias_ 2p4g is shown in Figure 5.11.

Table 5.1 Results of MCNC benchmarks of substrate sharing constraints

Circuit	Cells	groups		***Abs***	***S-TCG***	***S-TCG(m)***
apte	9	2	Cost	120.7%	50.74	48.67
			Time	1000%	3	4
ami33	33	2	Cost	118.9%	1.33	1.25
			Time	1670%	49	79
ami49	49	2	Cost	129.1%	42.21	41.03
			Time	2250%	86	134

Table 5.2 Results of Circuits of substrate sharing constraints

Circuit		*Abs*	*S-TCG*	*S-TCG(m)*
biasynth _2p4g	Cost	127%	5.42	5.27
	Time	2150%	107	152
lnamixbias_ 2p4g	Cost	119%	51.41	48.35
	Time (sec)	2430%	241	311
mod_ biasynth	Cost	123%	5.53	5.47
	Time (sec)	2137%	112	150
mod_lnamixbias_	Cost	120%	52.23	48.71
	Time (sec)	2440%	231	323
OTA	Cost	119%	27.53	25.25
	Time (sec)	2084%	223	327

Therefore, we can conclude that the proposed method is able to handle the substrate sharing constraint based on the TCG method, and it is efficient to enhance the cost performance of our TCG-based method.

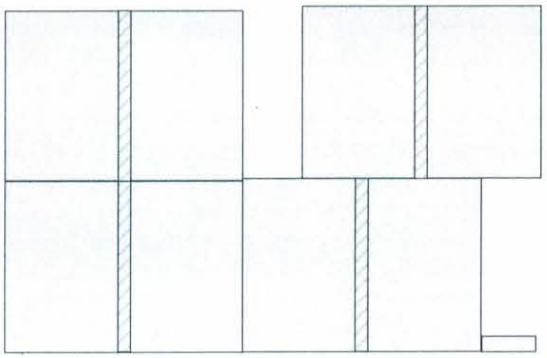


Figure 5.10 Placement of APTE

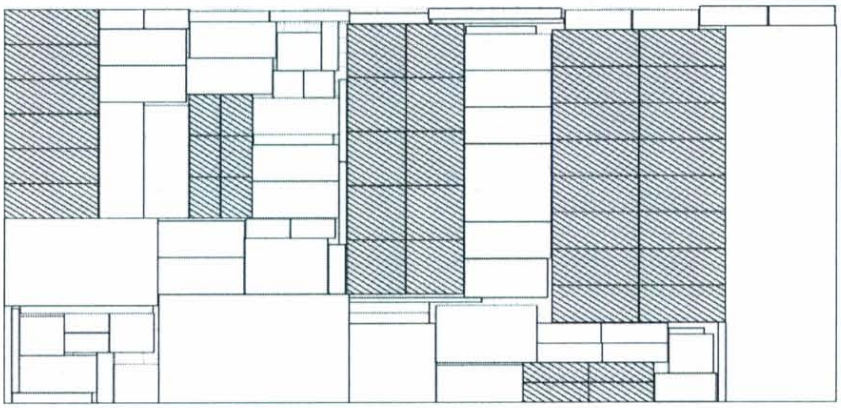


Figure 5.11 Placement of lnamixbias_ 2p4g

5.5 Other Constraints

Besides symmetry and substrate-sharing constraints, there are other constraints in the analog placement design such as relationship, abutment, and alignment constraints.

In this section, we briefly discuss the additional constraints that our proposed TCG-based placement scheme is able to handle. Due to the intrinsic features of the TCG representation, some other constraints, such as cell relationship, abutment, and alignment constraints, can be readily implemented in the current tool suite. Users are allowed to input multiple constraints based on some particular requirements of their desired placements.

5.5.1 Cell Relationship Constraints

In practical analog layout design, the designers may have specific placement requirement that demands one or more cells to have particular topological relationship with another cell. For instance, in Figure 5.12, one may require cell a to be located on the right of a symmetry group, which consists of cells b, c, d and e due to some sensitivity requirements. It is easy to handle this type of constraints when using TCG representation since we can just take them into account in the perturbation stage. In detail, we can input the relationship requirement at the initial stage so that cell a should be placed on the right of cell b. Then in the perturbation stage, we preserve the situation of $b \vdash a$ whenever changing the fan-in (fan-out) relationship of cell b or a in G_h. In this way, we can always guarantee this constraint to be observed in the following optimization.

However, for the tree-structure representations, such as HB*-tree, this operation

is nontrivial as there is no clue to reflect the relationship between each pair of cells based on the B*-tree representation. Furthermore, this constraint can be easily extended to require one symmetry group to have a particular relationship with another symmetry group. To realize it, we can simply add multiple constraints between the cells of one group and the cells of the other group. And one final placement of ami33 is shown in Figure 5.13.

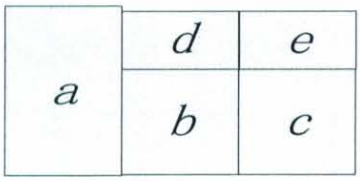


Figure 5.12 Example of relationship constraint

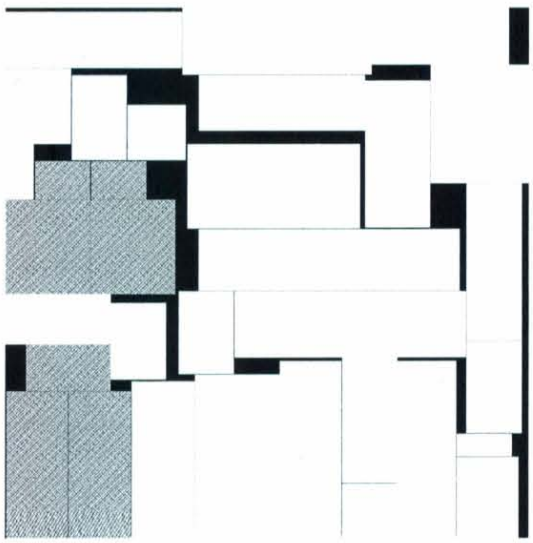


Figure 5.13 Example of ami33 with relationship constraints

5.5.2 Cell Proximity Constraints

Users can input the requirement of two cells that should be placed tightly. For example, in Figure 5.14, we may require that cell *a* is placed to vertically abut cell *b*, which forms a proximity placement. We can follow the concept of *island* employed in [10]. We first place the two cells abutting in the initial placement and then consider them as a connected island. In the TCG, the two vertices are combined together and treated as one dummy node. In other words, the two cells are always placed adjacent, and the weight between the cells is always the minimal value. And one final placement of ami33 with proximity constraints is shown in Figure 5.15.

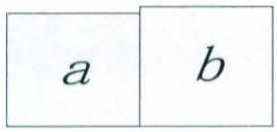


Figure 5.14 Example of proximity constraint

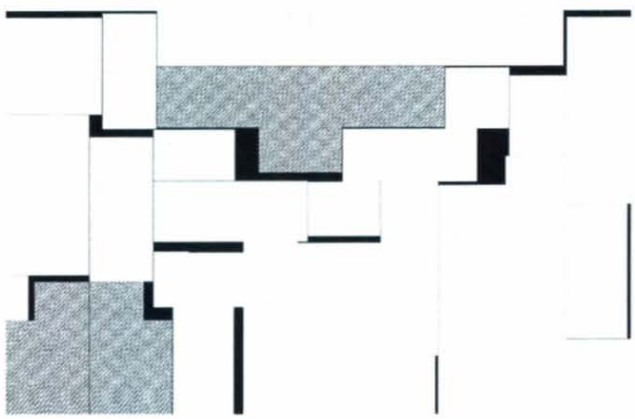


Figure 5.15 Example of ami33 with proximity constraints

5.5.3 Cell Alignment Constraints

We can also handle the cell alignment constraints that require one cell to be located vertically or horizontally in alignment with other cells. For this type of constraints, in the stage of initial placement, we first pack the cells following the topologic order one by one. If the current cell has an alignment constraint with another placed cell, we shift it to satisfy this constraint. Then it is considered as a pre-placed cell and the contour is updated accordingly in the final packing, which itself is the same as the packing scheme described in Chapter 4. If a symmetric cell has an alignment constraint, its symmetric counterpart should be shifted as well. As the example depicted in Figure 5.16, there is a vertical alignment constraint between the bottom sides of cells *c* and *b*. In contrast, it is impossible for the HB*-tree method to obtain this solution since the symmetric groups should be compactly packed and all the cells in one symmetry group should be always closely connected. And one final placement of ami33 is shown in Figure 5.17.

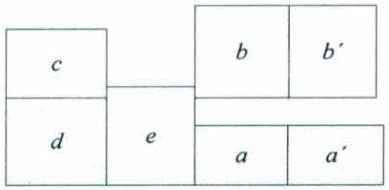


Figure 5.16 Alignment exmaple

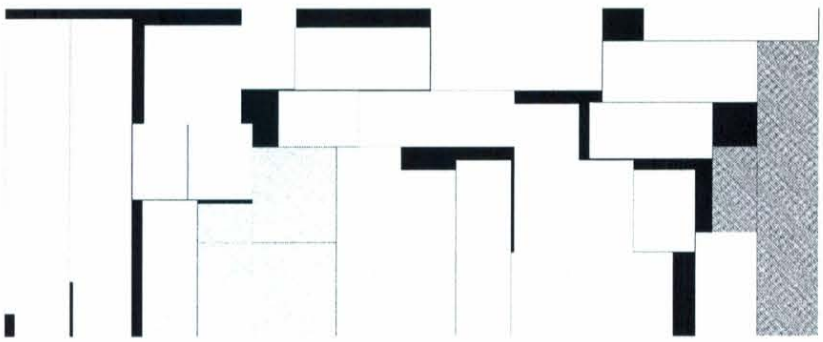


Figure 5.17 Example of ami33 with alignment constraints

5.6 Summary

In this chapter, I first reviewed the necessity of substrate-sharing constraints in the analog layout design. Then the formulation of the substrate-sharing constraints in our TCG-based placement scheme was discussed. Some experimental results were also provided to demonstrate the efficacy of our proposed approach. Finally the implementation schemes of other constraints, such as topological relationship, proximity, alignment constraints, were also discussed.

Chapter 6 Conclusions and Future Work

6.1 Conclusions

The main contributions presented in this thesis are the design, implementation, and performance evaluation of the TCG-based method to handle complex analog layout constraints including the multi-group symmetry, substrate-sharing and other topological constraints. The experimental results show that the proposed method works effectively and efficiently.

- After a thorough literature review on analog placement methods as well as distinct stochastic optimization techniques, I developed an artificial neural network based placement algorithm as a preliminary solution to the analog placement design problem. Some intrinsic drawbacks of this method, in particular, no guarantee of the final valid results as well as high complexity, have motivated me to investigate the topological placement methods.

- I carefully studied the features of different topological representations and tried to bridge the link between the topological representations and complex analog constraints. Among the different topological representations, I chose TCG

as the focus and developed TCG-based placement algorithms for handling complex analog constraints.

- For the symmetry constraints, I proposed several sufficient conditions to verify the symmetric feasibility of TCG representation to handle the multiple symmetry groups' situation. In addition, an efficient contour-based packing scheme and an *O(n)* perturbation scheme are introduced in our algorithm for the transfer between the TCG representation and placement. I have proved that the proposed algorithm can cover all the possible topological configurations of placement. And the experimental results show that this method achieves better performance compared with several recently published algorithms.

- For the substrate-sharing constraints, I propose a method to handle this type of constraints based on topological representation. The efficient packing and perturbation algorithm performs well and the results show that the area and wire length costs are noticeably decreased by applying this type of constraints. Moreover, our algorithm is easy to handle the relationship, proximity and alignment constraints due to the intrinsic features of the TCG representation.

6.2 Future Directions

Based on my current work reported in this thesis, several research directions can be expected. Besides the constraints covered in this thesis, there are still some other constraints that are helpful for the analog layout design. They are likely to be realized by extending the current TCG-based method proposed in this thesis. Moreover, the representation of TCG itself still has some room to be improved. In the following, these directions are listed in detail.

- Handling more analog placement constraints

More complex analog placement constraints include:

1) Common Centroid Constraints: The component mismatch has adverse effects on many analog circuits. One of the most important sources of mismatch is process gradient, like oxide thickness, threshold voltage, resistor layer thickness, etc. These kinds of mismatch can be effectively suppressed by common centroid layout, which refers to a layout style in which a set of devices have a common center point. Therefore, to reduce parasitic mismatch in analog circuits, some groups of devices may be required to share a common centroid while being placed [8]. Or devices may be split into a number of smaller ones, which can be placed with reference to the same center point.

2) Matching Constraints: The matching constraints force a common gate

orientation and an inter-digital placement among devices. It helps to reduce the effect of process-induced mismatches. The common centroid constraints are just one type of matching constraints.

3) Clustering constraints: The device constraints are usually specified according to circuit functionality in analog designs, such as current mirrors, differential pairs, and other sub-circuits. Besides the circuit functionality, constraints are also determined based on device model, substrate/well types or even designers' intents. If a set of devices belonging to a sub-circuit that needs to satisfy the matching, symmetry, or proximity constraint, they are usually formed as a device group or a cluster. Such clustering constraints can be hierarchically specified. That is, a cluster may contain not only device modules but also other clusters which contain other device modules or device groups. We can use a dummy node in the TCG to represent the clusters.

- Further study of the topological relationship

It can be observed that the sequence-pair, the hierarchical B*-tree and the transitive closure graph method all can only handle the horizontal and vertical relationships in the representations. The diagonal relationship in these representations is always ignored or classified as the horizontal or vertical relationship. As a matter of fact, recognizing the diagonal relationship is helpful when evaluating only the representation without packing. Therefore, it would be beneficial if one representation

can be found to handle the diagonal relationship. In this way, one is able to handle more complex analog placement constraints and speed up the evaluation or even the packing operations.

- Post-layout simulation of real analog circuits after the optimization

After applying our TCG-based method to handle complex analog placement constraints, it would be interesting to focus on not only the minimization of the area and wire length cost, but also the evaluation of performance of the analog circuits. This needs some additional work on layout routing and compaction. Then with post-layout simulation, one can understand and appreciate the significance of considering complex analog constraints in the placement stage.

References

[1] V. Beiu, J. M Quintana, and M. J. Avedillo, "VLSI implementations of threshold logic - a comprehensive survey," *Trans. of IEEE on Neural Networks*, vol. 14, pp. 1217-1243, 2003.

[2] G. Gielen and R. Rutenbar, "Computer-Aided Design of analog and mixed-Signal integrated circuits," *Proc. of the IEEE,* pp. 1825 – 1852, 2000

[3] E. Hjalmarson, "Studies on design automation of analog circuits- the design flow," *Ph.D thesis*, University of Linkopings, 2003.

[4] L. R. Carley, G. Gielen, R. Rutenbar and W. Sansen, "Synthesis tools for mixed-signal ICs: progress on frontend and backend strategies," *Proc. of IEEE/ACM Design Automation Conference*, pp. 298 -303, 1996.

[5] G. Gielen, *Analog circuit design*, Springer Netherlands, 2006.

[6] E. Martens and G. Gielen, "Classification of analog synthesis tools based on their architecture selection mechanisms," *the VLSI journal*, vol. 41, pp. 238-252, 2008

[7] R. Rutenbar, G. Gielen, and J. Roychowdhury, "Hierarchical modeling, optimization, and synthesis for system-level analog and RF designs," *Proc. of the IEEE*, vol. 95, pp. 640-669, June 2007.

[8] J. Cohn, D. Garrod, R. Rutenbar, and L. Charley, "Analog device-level automation," *Kluwer Academic Publishers*, 1994.

[9] P. Lin and S. Lin, "Analog placement based on symmetry-island formulation," *IEEE Trans. on CAD of integrated circuits and systems*, vol. 28, pp.791–804, June 2009.

[10] E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE Trans. on CAD*, vol. 15, pp. 923-942, Aug. 1996.

[11] J. Lin and Y. Chang, "TCG: a transitive closure graph based representation for general floorplans," *IEEE Trans. on VLSI Systems*, vol. 13, pp. 288-292, Feb. 2005.

[12] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI cell placement based on rectangle-packing by the sequence-pair," *IEEE Trans. on CAD,* vol. 15, pp. 1518-1524, Dec 1996.

[13] P. Guo, C. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," *Proc. of IEEE/ACM Design Automation Conference (DAC)*, pp. 268-273, 1999.

[14] Y. Chang, Y. Chang, G. Wu, and S. Wu. "B-tree: A new representation for non-slicing floorplans," *Proc. of IEEE/ACM Design Automation Conference (DAC)*, pp. 458-463, 2000.

[15] X. Hong, S. Dong, G. Huang, Y. Cai, C. Cheng, and J. Gu, "Corner block list representation and its application to flooplan optimization," *IEEE Trans. on Circuits and Systems II*, vol. 51, pp. 228-233, 2004.

[16] J. Lin and Y. Chang, "TCG-S: Orthogonal coupling of P*-admissible representations for general floorplans," *IEEE Trans. on CAD*, vol. 24, pp. 968-980, June 2004.

[17] P.-H. Lin and S.-C. Lin, "Analog Placement Based on Novel Symmetry-Island Formulation," *Proc. of IEEE/ACM Design Automation Conference*, pp. 465 – 470, 2007.

[18] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimizationby simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[19] D. Jepsen and C. Gelatt, Jr., "Macro placement by Monte Carlo annealing," *Proc. IEEE Int. Conf. Computer Design*, pp. 495–498, Nov. 1983.

[20] J. Cohn, D. Garrod, R. Rutenbar, and L. Carley, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 330–342, Mar. 1991.

[21] K. Lampaert, G. Gielen, and W. Sansen, "A performance-driven placement tool for analog integrated circuits," *IEEE J. Solid-State Circuits*, vol. 30, no. 7, pp. 773–780, Jul. 1995.

[22] W. Sun and C. Sechen, "Efficient and effective placement for very large circuits", *IEEE Trans. on Computer-Aided Design*, vol. 14, pp.349–359, 1995.

[23] F. Balasa and K. Lampaert, "Symmetry within the sequence-pair representation in the context of placement for analog design," *IEEE Trans. on CAD*, vol. 19, pp. 721-731, July 2000.

[24] Y. Tam, E. Young, and C. Chu, "Analog placement with symmetry and other placement constraints," *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pp. 349-354, 2006.

[25] L. Zhang, R. Raut, Y. Jiang, U. Kleine and Y. Kim, "A hybrid evolutionary analog module placement algorithm for integrated circuit layout designs", *International Journal of Circuit Theory and Applications,* vol. 33, pp.487–501, July 2005.

[26] S. Kouda, C. Kodama, and K. Fujiyoshi, "Improved method of cell placement with symmetry constraints for analog IC layout design," *Proc. of International Symposium on Physical Design,* pp 192-199, 2006.

[27] S. Kouda, C. Kodama, and K. Fujiyoshi, "Linear programming-based cell placement with symmetry constraints for analog IC layout," *IEEE Trans. on CAD of integrated circuis and system*. vol 26. pp 659-668, 2007.

[28] L. Zhang, R. Shi, and Y. Jiang, "Symmetry-aware placement with transitive closure graphs for analog layout design," *Proc. IEEE/ACM Asia and South Pacific Design Automation Conference,* pp 180-185, 2008.

[29] L. Zhang, N. Jangkrajarng, S. Bhattacharya, and R. Shi, "Parasitic-aware optimization and retargeting of analog layouts: a symbolic-template approach," *IEEE Trans. on CAD*, vol 5. pp 791-802, 2008.

[30] J. Liu, S. Dong, F. Chen, X. Hong, Y. Ma, and D. Long, "Symmetry constraint for analog layout with CBL representation," *Proc. of International Conference on Solid-State and Integrated Circuit Technology*, pp. 1760-1762, 2006.

[31] J. Lin, G. Wu, Y. Chang, and J. Chuang, "Placement with symmetry constraints for analog layout design using TCG-S," *Proc. IEEE/ACM Asia and South Pacific Design Automation Conference* , pp. 1135-1138, 2005.

[32] L. Zhang, R. Raut, Y. Jiang and U. Kleine, "Placement algorithm in analog-layout designs", *IEEE Trans. on CAD,* vol. 25, pp.1889–1903, 2006.

[33] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, pp. 1120–1135, Jun. 2003.

[34] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," *Proc. of ACM/IEEE Design Automation Conf.*, Las Vegas, NV, pp. 101–107, 1986.

[35] J. Cohoon, S. Hegde, N. Martin and S. Richards, "Distributed genetic algorithms for the floorplan design problem," *IEEE Trans. on CAD,* vol. 10, pp. 483-491, 1991.

[36] H. Kita, H. Odani and Y. Nishikawa, "Solving a placement problem by means of an analog neural network," *IEEE Trans. on Industrial Electronics,* vol. 39, no. 6, pp. 543-551, 1992.

[37] A. Gloria, P. Faraboschi and M. Olivieri, "Block placement with a boltzmann machine," *IEEE Trans. CAD,* vol. 13, pp. 694-701, 1994.

[38] L. Fang, W. Wilson and T. Li, "Mean field annealing neural net for quadratic assignment," *Proc. International Neural Network Conference*, pp. 282-286, 1990.

[39] M. Unaltuna and V. Pitchumani, "Quadrisectioning based placement with a normalized mean field network," *Proc. IEEE International Symposium on Circuits and Systems*, 1993, pp. 2047-2050.

[40] C. Petexson and J. Anderson, "Neural networks and NP-complete optimization problems," *Complex Systems,* vol. 2, pp. 59-89, 1988.

[41] X. Zhang and E. El-Masry, "A novel CMOS OTA based on body-driven MOSFETs and its applications in OTA-C filters", *IEEE Trans. on Circuits and Systems I*, vol. 54, no. 6, pp. 1204–1212, 2007.

[42] P. Khademsameni,. and M. Syrzycki, "A tool for automated analog CMOS layout module generation and placement", *Proc. IEEE Canadian Conference on Electrical and Computer Engineering*, pp 416-421, 2002.

[43]P.Drennan, M.Kniffin and D.Locascio, "Implications of Proximity Effects for Analog Design". *Proc. IEEE Custom Integrated Circuits Conference,* pp. 169-176, Sept. 2006.

[44]P. H. Lin and S. C. Lin, "Analog Placement Based on Hierarchical Module Clustering," *Proc. Design Automation Conference*, Anaheim, California, USA, pp. 50-55, June 2007.

Appendix - List of Publications

1. ***Rui He*** and L. Zhang, "Analog Symmetry Placement Design Using Transitive Closure Graphs," *Proc. IEEE Newfoundland Electrical and Computer Engineering Conference*, Nov. 2008.

2. ***Rui He*** and L. Zhang, "Evaluation of modern MOSFET models for bulk-driven applications", *Proc. of IEEE Midwest Symposium on Circuits and Systems*, pp. 105-108, Knoxville, U.S.A., 2008.

3. ***Rui He*** and L. Zhang, "Analog placement design with constraints of multiple symmetry groups", *Proc. of IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 1204- 1207, 2009.

4. ***Rui He*** and L. Zhang, "Artificial neural network application in analog layout placement design", *Proc. of IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 954- 957, 2009.

5. Shaoxi Wang, ***Rui He*** and L. Zhang, "MOSFET model assessment for submicron and nanometer bulk-driven applications", *Proc. of IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 954- 957, 2009.

6. ***Rui He*** and L. Zhang, "Symmetry-Aware tCG-Based placement design under

complex multi-group constraints for analog circuit layouts", *Proc. IEEE/ACM 15th Asia and South Pacific Design Automation Conference*, pp. 299–304, 2010.

7. ***Rui He*** and L. Zhang, "Symmetry-Aware Analog Layout Placement Design Handling Substrate-Sharing Constraints," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 2398–2401, 2010.