An Advanced Graph-Based Placement Representation for Analog Layout Design

by

© Lian He

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

Department of Computer Science
Faculty of Science
Memorial University of Newfoundland

May 2021

St. John's, Newfoundland and Labrador

ABSTRACT

Due to complexity and susceptibility of analog layouts towards circuit performance, maturity state of analog integrated circuit (IC) physical design automation has largely lagged behind that of the digital counterpart. Placement is an indispensable stage in the analog IC layout design. It demands effective representations to handle nontrivial analog placement topologies especially in the advanced nanometer technologies. In this thesis, we mainly review the existing placement representations and deepen the research of topological representations for the analog placement design.

By leveraging the equivalence between sequence pair (SP) and transitive closure graph (TCG), we propose an SP-driven TCG representation and its associated operations to facilitate the handling of analog placement constraints. To achieve the symmetry-aware placement, we introduce a set of special symmetric-feasible conditions and define an efficient construction mechanism for symmetric placement with the SP-driven TCG representation. A set of SP-driven perturbation operations is also brought forth in this thesis to reduce the algorithmic complexity while satisfying symmetry constraints. Furthermore, a redundancy control scheme among the representation states is developed in order to generate high-performance analog placement with high computation efficiency.

Based on the SP-driven TCG representation, we further introduce an Advanced Transitive-Closure-Graph-based placement representation (ATCG). It can effectively and efficiently tackle advanced geometric constraints, which are highly essential for addressing

layout dependent effects, thermal effects, and diverse parasitic challenges in the advanced nanometer technologies. ATCG not only inherits all the advantage from both SP and TCG, but also resolve the ambiguous diagonal relationship between any two specified modules. The versatility and flexibility of ATCG can ensure it to accurately control spacing and merging constraints uniquely required by analog layout design. We have implemented our proposed placement methods and tested them with several circuits. Our experimental results demonstrate high efficacy of these proposed representations and the developed operations.

ACKNOWLEDGEMENTS

I would first like to convey my gratitude to my supervisor, Dr. Yuanzhu Chen, for his continuous encouragement, support and patience throughout my years of research study. He consistently guided me to focus on interesting topics in the research. Whenever I encountered any difficulty, he was always there for help. Without his assistance, I was not able to succeed in conducting this research work. Moreover, I would like to thank Memorial University and Department of Computer Science for offering me this precious opportunity to pursue my postgraduate studies.

I would deeply appreciate being advised by Dr. Lihong Zhang. He always offered his valuable suggestions during my research project. His expertise in the field of VLSI and EDA helped me expand my knowledge and open my mind. I could discover new things from every discussion with him.

This thesis would not have been possible without graduate study support and research facilities provided by the School of Graduate Studies, the Faculty of Science and the Faculty of Engineering & Applied Science at Memorial University.

I am also grateful to the graduate student fellows for their kind help, discussions, and suggestions.

Finally, I must express my special thanks to my family who give me all their unconditional love and care all the time.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ASF : Automatically Symmetric Feasible

ATCG : Advanced Transitive Closure Graph

BSG : Bounded Slicing Grid

CBL : Corner Block List

CB-tree : Corner Stitching Compliant B*-Tree

CMOS : Complementary Metal–Oxide–Semiconductor

CNTFET : Carbon Nanotube Field-Effect Transistor

EDA : Electronic Design Automation

FinFET : Fin Field Effect Transistors

GA : Genetic Algorithm

HB-trees* : Hierarchical B*-Trees

IC : Integrated Circuits

IoT : Internet of Things

LDE : Layout Dependent Effects

LP : Linear Programming

MOM : Metal-Oxide-Metal

MOSFET : Metal Oxide Semiconductor Field-Effect Transistor

NPE : Normalized Polish Expression

Q.E.D. : Latin phrase "Quod Erat Demonstrandum" which means "that which was to be demonstrated". A notation is often placed at the end of a mathematical proof to indicate its completion.

SA : Simulated Annealing

SoC : System-on-Chip

SP : Sequence Pair

STL : Standard Template Library

TCG : Transitive Closure Graph

VLSI : Very Large Scale Integration

List of Symbols

α :	α-sequence of Sequence Pairs
β :	β-sequence of Sequence Pairs
E :	Edge
G :	Graph
h_i :	Height of Module i
H_{atcg} :	Horizontal Advanced Transitive Closure Graph
H_{tcg} :	Horizontal Transitive Closure Graph
m_s :	Self-Symmetric Module
m_p :	Representative of Symmetric-Pair Modules
p_i :	Representative of the i^{th} Symmetric-Pair Modules
s_i :	i^{th} Self-Symmetric Module
V :	Vertex
V_{atcg} :	Vertical Advanced Transitive Closure Graph
V_{tcg} :	Vertical Transitive Closure Graph
w_i :	Width of Module i
w_{ij} :	Edge Weight from Edge i to Edge j

1. Introduction

The invention of integrated circuit (IC) has brought about a revolution in our modern world. ICs are essential components of virtually all electronic equipment, from the space rocket to personal handheld devices. As very-large-scale-integration (VLSI) technology continues to advance, mixed-signal integrated circuits, i.e., integration of analog and digital circuits, are now ubiquitous in our daily life through emerging Internet of Things (IoT). Electronic design automation (EDA) tools greatly facilitate IC designers in their sophisticated design process.

An essential stage in physical design is placement/floorplanning, whose objective is to optimally place assorted modules on a chip according to the design requirements. A placement representation is used to describe the topological relationship among modules. Due to complex analog layout constraints, commercial readiness of analog design automation tools always lags behind that of the digital counterpart. In particular, the emergence of layout dependent effects (LDE) [1][2] in the nanometer technologies demands a proper placement representation, which is applicable to susceptible analog placement topologies.

Researchers have been continuously developing various representations to meet the placement requirements in IC layout design. Generally, the placement representations can be classified into *absolute representation* and *topological representation*. The *absolute representation*, also called Jepsen–Gelatt style flat representation [3], is based on absolute coordinates on a gridless plane. A module can be manipulated by directly changing its

coordinates. This straightforward representation was widely used in all early layout systems, not only in digital design (e.g., TimberWolf [4]), but also in analog design (e.g., KOAN/ANAGRAM II [5], LAYLA [6], and ALADIN [7][8]). It was normally considered as a less effective representation due to extremely large configuration space associated. Lately, Ou *et al*. proposed an LDE-aware analytical analog placement model by returning to this representation [9]. However, since geometry overlaps are inevitable in the placement solutions, a detailed placement process has to be resorted, which means both computational efficiency and solution optimality are compromised.

Many *topological representations* have been developed for placement design in the recent years. They can be categorized into *slicing* and *non-slicing topological representations*. A *slicing topological representation*, e.g., slicing tree [10] or normalized Polish expression [11], is derived from a slicing placement, which can be produced by recursively bisecting the layout horizontally or vertically into smaller modules [12]. Although the slicing representation has such superiority as smaller solution space and faster run time, it may lead to low performance for analog placement since most of analog layouts are non-slicing. Therefore, the researchers have been diligently exploring different *non-slicing topological representations* for analog placement design, e.g., sequence pair (SP) [13], bounded-slicing grid (BSG) [14], O-tree [15], B*-tree [16], corner block list (CBL) [17], transitive closure graph (TCG) [18], TCG-S [19], HB*-trees [20], QB-trees [21], and WB-trees [22].

1.1 Motivation

Complementary metal–oxide–semiconductor (CMOS) is widely used in digital, analog and mixed-signal ICs, such as microprocessors, image sensors, data converters, and highly integrated wireless transceivers, due to its high noise immunity and low static power consumption. CMOS is a combination of two types of metal–oxide–semiconductor field-effect transistors (MOSFETs), i.e., NMOS and PMOS, which are illustrated in Figure 1.

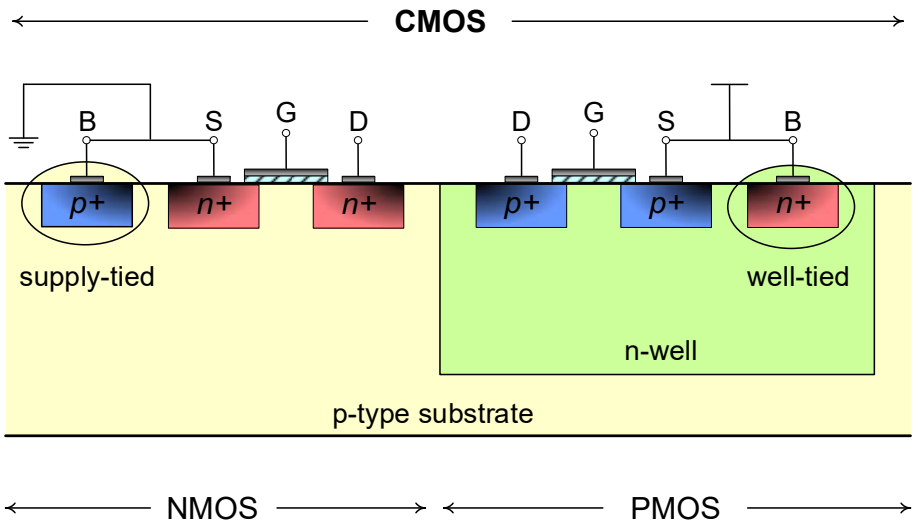


Figure 1. Complementary metal–oxide–semiconductor.

Figure 1 shows that both NMOS and PMOS transistors have four terminals: *source (S), drain (D), gate (G)* and *bulk (B)*. The source and drain terminals in an NMOS transistor occupy two individual deeply doped n^+ areas, which are built in a *p-type substrate*. On the contrary, the source and drain terminals with their individual deeply doped p^+ regions are formed in an *n-well* for a PMOS transistor, which is also built in the same p-type substrate.

Between the source and drain terminals, there exists the gate terminal, which is a conductive material to cover a thin insulation layer usually made of silicon dioxide. Previously polysilicon was applied as gate material in MOS devices, while metal gate is preferred at present. Generally, we call well or substrate region as the bulk terminal.

The right part of Figures 2 and 3 illustrates the patterns of NMOS and PMOS devices laid out in EDA tools [23]. In Figure 2, the source and drain regions are formed by Diffusion layer (or called Active layer in some technology processes) enclosed by NPLUS layer, while the bulk region is formed by Diffusion layer but enclosed by PPLUS layer. In Figure 3, the source and drain regions are formed by Diffusion layer enclosed by PPLUS layer, while the bulk region is formed by Diffusion layer but enclosed by NPLUS layer. All of the source, drain, gate, and bulk regions in the PMOS transistor are enclosed by NWELL layer.

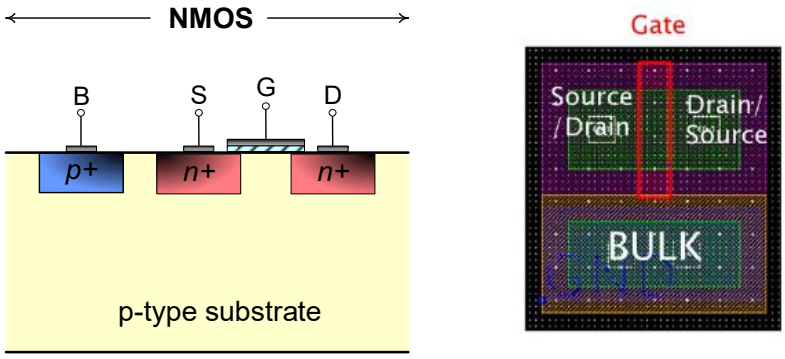


Figure 2. NMOS device.

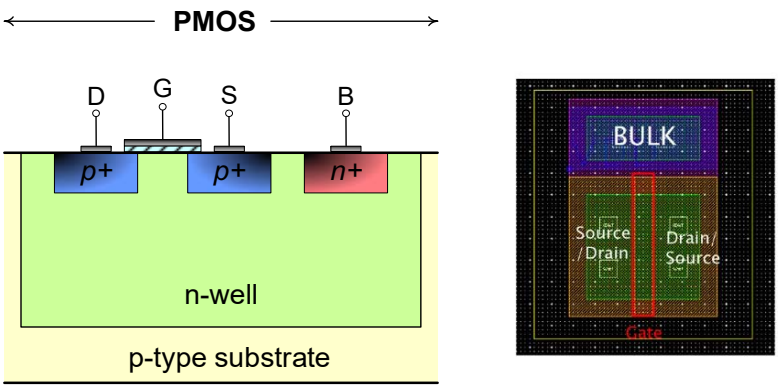


Figure 3. PMOS device.

In analog design, the well of a MOSFET (in particular, a critical transistor in terms of circuit performance) is typically tied to its own source rather than the common well bias potential through bulk contacts. The purpose of this practice is to reduce the threshold voltage shift due to the body effect. This kind of devices are thus termed as *well-tied devices.* They may seriously affect the density of layout since they must be spaced with a considerable distance away from other wells [24]. In addition, Figures 4 and 5 show that analog and digital devices are commonly required to be separated from each other in analog and mixed-signal design [25][26]. Even in the modern design, these requirements still exist as shown in the right part of Figure 5. Moreover, LDEs, optical proximity correction and thermal effects in analog design also require spacing constraints.

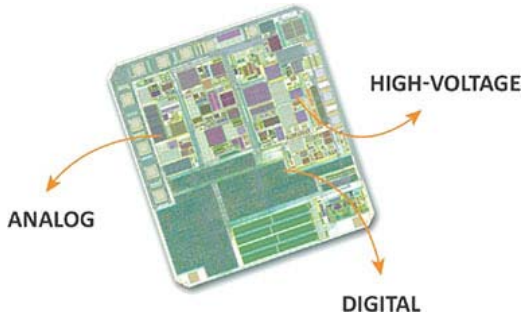


Figure 4. Mixed-signal integrated circuits.

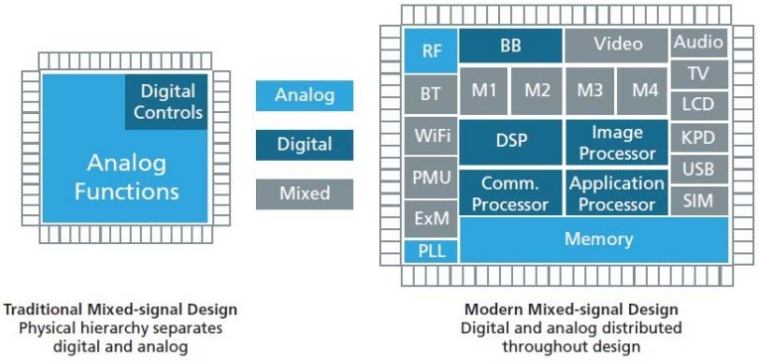


Figure 5. Mixed-signal designs.

In contrast, for saving silicon area it is normally preferred to merge substrate/well contacts of the MOS devices in the layout if their bulk terminals are biased at the same supply potential. If the sources of these devices are connected to appropriate supply, i.e., GND or VDD, we call them *supply-tied devices* [24]. For the supply-tied MOS devices that are supposed to share the same potential for their bulk terminals, their substrate/well regions can be merged as well.

In this thesis, we propose an advanced graph-based placement representation, called *advanced transitive closure graph (ATCG, for short)*. Our motivation of developing ATCG is to achieve constrained placements, that is, keeping space between modules or merging modules. More details of our proposed ATCG method, including SP-driven TCG approach, are discussed in Chapter 3 and Chapter 4. It can be included in EDA software to facilitate IC design tasks.

1.2 Organization of the Thesis

The thesis is organized as follows. Chapter 1 gives a brief introduction of analog layout design automation and development of placement representations. Moreover, the motivation of this thesis work is presented.

Chapter 2 reviews the previous work on the general placement methods and the categorized placement representations. After elaborating on the analog placement problem, we focus on a survey of different analog placement techniques that have been published over the time in the literature. Subsequently, we discuss the advantages and weaknesses of these techniques.

Chapter 3 defines the symmetric-feasible conditions and presents the transformation between symmetric placement and symmetry-aware TCG. A detailed SP-driven

perturbation scheme and its redundancy control are introduced. Our proposed approach is compared with the other methods and its effectiveness is also demonstrated.

In Chapter 4, we proposes an innovative ATCG placement representation to facilitate the handling of analog layout design. We detail the transformation between a placement and its corresponding ATCG. The properties and validity of ATCG are theoretically proved. After that, our proposed ATCG placement handling mechanism is presented in detail. Its superiority for analog layout design is verified through our experiments.

Finally, we draw conclusions and discuss the future scope of this work in Chapter 5.

2. Previous Work on Placement Methods

In this chapter, different general placement methods and representations are firstly reviewed. After that, we explain why it is challenging to deal with the analog placement problem. Then, we survey the techniques that have been utilized in the literature to handle analog placement.

2.1. Previous Work on General Placement Methods and Representations

2.1.1. General Placement Methods

After over half a century of development, VLSI technology is now penetrating more deeply and broadly into our everyday modern life. To address the complexity and improve the productivity, EDA has become an essential aid to IC designers. As a significant step in physical design EDA, IC placement, which is also called floorplanning, aims to place assorted circuit components within a chip according to the design requirements. The principal objective of the placement problem is to locate a set of non-overlapping modules optimally on a chip. The total area of the chip should be minimized along with the consideration of certain constraints, such as minimal wirelength, balanced channel density, satisfactory manufacturability, etc. Not only does an inferior placement greatly affect the performance of a chip, but also might make the design non-manufacturable.

In VLSI layout design, the placement problem of an arbitrary number of functional rectangular or rectilinear modules/cells consists of two basic NP-hard problems [24], namely, the chip size minimization problem and the total virtual wirelength minimization problem. Owing to the complexity of the placement problem, several types of approaches have been utilized. They can be typically categorized into constructive approaches, branch-and-bound approaches, analytical approaches, min-cut approaches, iterative approaches, etc.

Constructive placement methods were among the earlier developed placement methods for VLSI layout [27]. The placement result is acquired by choosing only one module at a time and then locating it in the relatively best available position. These approaches not only strictly depend on the selection order of modules, but also neglect global views. Therefore, they might lead to poor solutions, even though they are generally fast.

Branch-and-bound placement methods search all possible layout space using a form of controlled enumeration. The search range might be reduced by a lower bound calculation of the objective function. Since the amount of the visited configurations grows exponentially with the size of the problem, these schemes are applicable only to the small scale problems. Thus, they have been generally applied to find the optimal solutions to the placement problems, which only include a very limited number of modules [27].

Since *analytical methods* were introduced in the 1970's, they have become one of the most common techniques in handling the placement problem. Analytical placement algorithms ordinarily utilize a quadratic or nonlinear wirelength objective function [28].

The main advantage of these methods is that the placement problem is transformed into mathematical formulation, which can be resolved with efficient mathematical solvers. Although analytical techniques are relatively efficient in dealing with large problems and have a global view of the placement problem, they are not able to cope with the placement with some special constraints.

Min-cut placers use graph partitioning algorithms to set the modules/cells [29]. The basic partitioning is bisection, that is, the circuit is partitioned into two parts. The available layout area is also divided into two sections by a horizontal/vertical straight cutting line. Then each of the circuit partitions is assigned into one of the sections. Thus, the original placement problem is split into two smaller sub-problems. This recursive process of partitioning and solving is terminated until each section consists of a few or even only one cell. Since partitioning-based approaches do not directly attempt to minimize wirelength, the solution obtained by these methods is sub-optimal in term of wirelength.

Good quality placements are typically generated by *iterative methods*, although a relatively large amount of time is required. Given an initial configuration, iterative techniques randomly pick pairs of modules and exchange them. This interchange is accepted if it leads to a reduced cost. The algorithm keeps on searching within the given time until no better available results can be found. *Simulated annealing (SA)* [3] is one of the most popular iterative algorithms. Another one is *genetic algorithm (GA)* [4], whose iterative process is in a similar way, but with different initial configurations.

The concept of SA was originated from the operation of annealing metal, which is a heating and cooling process to alter the metal structure and make the material more robust.

SA sets a variable as temperature in order to simulate this heating and cooling treatment. The variable is initialized to a positive value and then gradually decreased to zero, which helps implement the slow cooling process. During the execution, a random solution is produced on each iteration and compared with the current solution. When the temperature is relatively high, worse solution may be accepted to avoid only searching and being entrapped to any local optimum solutions. The possibility of accepting worse solutions is gradually reduced as the temperature decreases. In this way, SA can effectively find a global optimum solution even when coping with the problems that contain numerous local optimums.

The key idea of GA is from "survival of the fittest", Charles Darwin's theory of evolution by natural selection. GA starts with a population consisting of a group of individuals, which are some solutions to the problem you need to solve. A fitness function is used to calculate the fitness value for each individual. The individuals with higher value are selected as the parents and then produce their offspring, who inherit the characteristics of the parents. Genetic operators, such as crossover, mutation, and inversion, are normally utilized during the production. The new generation is expected to have better quality than its parents. This selection process continues until a generation comprised of the best individuals is found.

In general, both SA and GA have good searching capability, while SA is deemed as the faster one regarding the execution time [30]. SA and GA are also classified as *stochastic approaches*. This kind of algorithms may produce a different result after every run. In contrast, the algorithms based on formulas or mathematical models are called deterministic,

since they can always output the same solution to a specific placement problem [31]. *Deterministic approaches* usually include constructive techniques, branch-and-bound techniques, and analytical techniques.

2.1.2. Placement Representations

A placement algorithm generally works with a certain representation in order to effectively handle the placement, which features various constraints. In the literature, the placement representations are classified into *absolute representations and topological representations*.

Absolute Representations

An absolute representation, also called Jepsen–Gelatt style flat representation [3], is based on absolute coordinates on a gridless plane. A module/cell can be manipulated by directly changing its coordinates. This straightforward representation was widely used in all early layout systems, not only in digital design, such as TimberWolf [4], but also in analog design, e.g., KOAN/ANAGRAM II [5], PUPPY-A [32], and LAYLA [33]. However, overlaps may be found in the final placement solution and thus additional processes are still needed to handle the overlap situation, which means computation time and solution optimality might be inevitably degraded.

Topological Representations

Increasingly topological representations have been applied in placement design in the recent two decades. A topological representation was derived from a slicing placement, which can be produced by recursively bisecting the layout horizontally or vertically into smaller modules [12]. A binary-tree, or called slicing tree, is used for the representation of the slicing placement [10]. A couple of years later, Wong and Liu proposed a normalized Polish expression (NPE) to represent a slicing placement [11]. Although the slicing representation has such superiority as smaller solution space and faster run time, it degrades layout density, especially for analog layout. In addition, most of the analog layout placements are non-slicing in the real world anyway. Therefore, in the recent years, the researchers have been paying more attention to non-slicing topological representations, e.g., SP, BSG, O-tree, B*-tree, MB*-tree, CBL, TCG, TCG-S, HB*-trees, QB-trees, and WB-trees.

SP is regarded as a flexible representation for general placement. It was proposed by Mutara *et al.* [13], who recommended using two sequences of module permutations to encode the left-right and up-down relations between modules. Since the location relationship including distance between modules is not transparent to the operations of SP, constraint graphs are needed to be constructed from scratch for SP cost evaluation after every perturbation. This is time-consuming and makes SP inconvenient to handle the placement problems with certain constraints, e.g., boundary modules, preplaced modules, range constraints, etc.

The *BSG representation* was introduced by Nakatake *et al*. [14], who suggested to use a meta-grid structure without physical dimensions to define the orthogonal relationship between modules. Many redundancies exist because there could be quite a few different representations corresponding to one placement. Consequently, BSG needs a larger solution space and longer search time to identify an optimal solution.

Tree-based representations are available to lessen the redundancies in the SP and BSG representations. They utilize ordered tree to represent compacted placement. The O-tree representation was proposed by Guo *et al*. [15] and a binary tree-based representation, namely B*-tree, was introduced by Chang *et al*. [16]. Every node of a tree corresponds to a module of a compacted placement. The root node denotes the bottom-left module. An edge in an O-tree represents the horizontal relations of two adjacent modules. In a B*-tree, the left child of a node indicates a right-left relationship between two abutting modules, while the right child represents an above-below relationship. A representation called MB*-tree was developed by Lee *et al*. [34], who proposed an agglomerative multilevel placement framework. A two-stage technique, i.e., clustering followed by declustering, is employed to handle complex placement. The advantages of the tree-based representations are the reduced configuration space and fast packing time. However, they can only represent compacted placements. Since certain optimal solution might result from uncompacted placements, these tree-based representations might be only able to find suboptimal solutions per se.

The *CBL representation* was proposed by Hong *et al*. [17]. They defined a corner block list to represent mosaic placements. A CBL is a three tuple (S, L, T), where S is a

sequence of module names, L is a list of module orientations, and T is a list of T-junction information. It takes linear time to switch a CBL to its corresponding placement. It is also available to handle several placement constraints, such as boundary constraints, abutment constraints, etc. However, there is no guarantee of obtaining a feasible solution after each perturbation. Since CBL can only represent placements without empty rooms, the optimum placement is hard to be generated if any separation constraint exists.

In a *TCG representation*, which was first introduced to the EDA area as a promising placement solution by Lin and Chang [18], two transitive closure graphs are used to represent the horizontal and vertical relationships between each two modules, respectively. TCG is able to support incremental update and hold such information as boundary modules, layout shape, spacing relations among modules. The most important strength of TCG lies in the fact that the geometric relationship of modules is transparent throughout the graph operation. This advantage is highly helpful for handling placement with complex constraints. Since TCG bears a large number of edges, which are needed to be validated during perturbation, a high computation complexity is inevitable. In addition, Lin and Chang proposed another graph-based representation, called *TCG-S* [19], which integrates TCG with part of SP. The packing time is shortened by using a packing sequence. TCG-S retains the advantages of TCG and combines them with the desired features of SP. Nevertheless, validity checking of edges increases the time complexity of the algorithm.

2.2. Analog Placement Techniques

2.2.1 Analog Placement Problem

Consumer telecommunication systems and wireless communication devices, such as cell phones and wireless local area network systems, require high-performance analog IC. Currently mixed-signal ICs, i.e., integration of analog and digital circuits together, are pervasive in our real world. The typical examples include system-on-chip (SoC) devices, cellular, Bluetooth, etc. Compared with well-developed digital design automation, automated design for analog circuits actually far lags behind [35]. Although analog circuits normally contain only a small number of devices, they are most sensitive to layout parasitics, and the inherent analog layout requirements of symmetry and matching make a unique challenge to analog designers. In the design of such analog IC, people have to take into account the balance of layout-induced parasitic effects to avoid performance degradation, since analog circuits are very sensitive to parasitic disturbance, crosstalk, and power supply noise.

The ideal tool for analog IC designers should smartly explore various floorplans and then instantly produce a best response for the subsequent verification. So far, all automation attempts have still been struggling with generating high qualified analog layouts [36]. EDA developers have tried to solve the analog layout generation problem by use of digital placement techniques. However, it seems hard to satisfy analog designers especially in terms of special analog constraints. Since most of the analog circuits are quite distinct from

one another based on their various applications, it is still too complicated to be managed by the existing EDA tools. Therefore, automated analog layout design has been always stubborn work in the past decades and analog designers tend to manually lay out their circuits to seriously take into account various constraints. However, this is an error-prone and time-consuming process that suffers from a lot of productivity problems.

Meanwhile, technology evolution towards nanometer era has imposed many advanced constraints on analog layout design, such as various layout dependent effects (LDE) [1][2]. Hence, its high complexity demands a proper representation to deal with susceptible analog placement configurations and nontrivial constraints. The traditional analog layout constraints include symmetry, common-centroid, and proximity constraints [37], among others.

The layout design of differential structure always demands to consider symmetry constraints, which help suppress the parasitic mismatch and thermal effect between two identical signal flows in the differential modules. Normally a current-mirror or differential-pair module needs to apply common-centroid constraints to reduce process-induced mismatch among the devices. The proximity constraints are usually required in the structure that needs specified distances among enclosed components. They help form a connected placement of modules so that the modules can share a connected substrate or well region to reduce the layout area, the interconnecting wire length, and the substrate coupling effect. The placement outline of modules with such constraints can be irregularly rectilinear in order to properly utilize silicon area.

2.2.2 Analog Placement Techniques for Symmetry Constraints

For optimal analog placement design, researchers have been consecutively developing many techniques using various topological representations to meet the primary placement requirements, e.g., symmetry constraints. Normally, they use an SA algorithm to perturb placement configurations while satisfying symmetry constraints [39].

For the first time, Balasa and Lampaert [40] explored the SP representation in the context of symmetry placement for analog layout design and derived the symmetric-feasible conditions in SP. This is the first work on symmetry-aware topological placement study, which triggered the intensive research in this area for the subsequent decade. After several years, Koda *et al*. found a few intrinsic defects in this method, such as incorrect necessary condition, overlapping modules, etc. [41]. To address these issues, they proposed an approach based on linear programming (LP) by using "symmetric-real-feasible SP", where linear expressions are employed to shorten the running time. Nevertheless, high packing complexity is inevitable due to the nature of LP.

A few tree-based representations, such as O-tree and B*-tree, have been also applied to reduce high packing complexity for achieving analog symmetric placement. The O-tree representation was proposed by Pang *et al*. [42], who introduced symmetric X-feasible and Y-feasible conditions to tackle the symmetry constraints. However, the feasibility of the O-tree solutions can only be detected after packing, which means they have to explore the whole configuration space to find feasible solutions with the symmetry constraints. Thus, this work suffers from longer run time. An augmented B*-tree representation was proposed

by Balasa *et al*. to handle the symmetry constraints [43] by means of symmetric-feasible conditions on B*-tree. The symmetric-feasible B*-tree is efficiently assessed by using a segment tree, which is a data structure normally deployed in computational geometry.

A hierarchical B*-trees (HB*-trees) was proposed by Lin *et al*. [20], who introduced a concept of symmetry island to solve symmetry constraints. A symmetry island is formed by modules of the same symmetry group in a single connected placement. Based on this concept and the B*-tree representation, an automatically symmetric-feasible (ASF) B*-trees is used to directly model the placement of a symmetry island. Then HB*-trees are formed to simultaneously optimize the placement with both symmetry islands and non-symmetric modules. This work tends to be deemed as one of the best performed analog placers in the past two decades. Nevertheless, even though this is the first approach that can handle the placement with symmetry constraints in linear time, it can only cope with the situation where symmetric modules belonging to the same symmetry group are closely adjacent to each other, i.e., no asymmetric module among the symmetric ones.

The TCG and TCG-S representations have been also utilized to deal with the analog placement considering symmetry constraints. Zhang *et al.* proposed TCG-based symmetric-feasible conditions for implementing analog symmetric placements [44]. However, this work is only limited to the analog circuits that feature one symmetry group. Later He and Zhang presented a complete set of symmetric-feasible conditions to handle the multiple symmetry constraints for analog layout placement [45]. Although perturbation time complexity is improved to linear time, the symmetric packing operation tends to be too complicated and heuristic. Lin *et al.* introduced TCG-S representation to realize the

symmetric placement [46], but the proposed symmetry-feasible conditions are not applicable to every possible symmetric situation. Moreover, the symmetric modules fail to include self-symmetric ones and the proposed perturbation scheme cannot guarantee the promised symmetric feasibility.

2.2.3 Analog Placement Techniques for Other Constraints

Analog placement with symmetry constraints has been extensively studied in the literature. Although people have extended their scope to other analog constraints, so far there are only a few previous works that can simultaneously consider symmetry constraints along with other placement constraints.

In [47] Tam *et al*. proposed to utilize SP as the representation in the SA engine to handle the following placement constraints: device separation constraint, alignment constraint, abutment constraint, boundary constraint, preplaced constraints and range constraint besides symmetry constraint. Augmented constraint graphs are deployed to satisfy all the placement and symmetry constraints at the same time by adjusting the edge weights. Since Balasa's method [40] is adopted, the inherent problems in their SP symmetric-feasible conditions are inevitable.

Another approach was proposed by Xiao and Young [48], who defined their own feasibility conditions also based on SP representation. These conditions are applied to generate the placements satisfying the common centroid and 1-D symmetry constraints.

Their developed condition definitions are based on the concept of group clustered placements, which is similar to HB*-trees.

Ma *et al*. [49] improved the work of Tam *et al*. [47]. They presented a placement methodology that can simultaneously handle symmetry constraints, common centroid constraints, and other general placement constraints. C-CBL representation is utilized to represent the placement of a common centroid group, which is treated as a super-block in SP. However, the intrinsic problems still exist as in [47].

Recently, Ou *et al.* proposed an LDE-aware analytical analog placement model [9] by returning to the traditional absolute coordinate representation in order to deal with those advanced proximity and spacing constraints. However, since geometry overlaps are inevitable in the initial global placement solutions, a complex overlap removal scheme has to be resorted to in the subsequent detailed placement process. That means both computational efficiency and solution optimality are unfortunately compromised.

Although the B*-tree representation has shown to be highly effective and efficient for floorplan/placement problems, it has an intrinsic limitation in deriving module adjacency information directly from the representation itself. To address this issue, Tsao *et al.* presented a corner stitching compliant B*-tree (CB-tree, for short) to remedy the significant deficiency in its module adjacency handling [50]. A CB-tree is a B*-tree integrated with modified corner stitching to offer much higher flexibility/efficiency, especially for adjacent module identification/packing. Wu *et al.* proposed a representation combining a quadtree data structure and the B*-tree representation, i.e., QB-trees [21]. They presented a comprehensive system with the hybrid representation to handle all the

general geometrical constraints, while achieving lower-bound time complexity of module packing and constraint handling. But its core scheme of symmetry handling still follows HB*-trees [20].

To assist in Fin Field Effect Transistors (FinFET)-based analog layout designs, Wu *et al.* presented a scheme for parasitic-aware common-centroid placement and routing dedicated to current-ratio matching [51]. Although this is among the earliest works on FinFET layout automation in the literature, the proposed method itself is only limited to analytical modeling and formulation. Instead of using any common EDA tools, a Matlab toolbox was deployed for verification purpose in the experiments. Recently, Lu *et al.* proposed WB-trees representation [22], which is a hybrid of window mesh data structure and the CB-tree representation. They presented how to handle FinFET-induced constraints, such as fin alignment, mask conflict, and mask density balance.

In the recent years, some research activities have been also reported in the literature to study the placement and routing strategies for special analog and mixed-signal circuits or structures. For instance, Lin *et al.* proposed a method for parasitic-aware common centroid binary-weighted capacitor layout generation [52]. It includes the consideration of placement, routing, and unit capacitor sizing. Such generated capacitors (e.g., metal-oxide-metal (MOM) capacitors) can be used in switched capacitor circuits, digital to analog converters, analog to digital converters, etc. [53].

2.3. Summary

In this chapter we reviewed different general placement methods and representations. Then we introduced the analog placement problem and surveyed various techniques for handling analog placement. We also discussed the advantages and limitations of these techniques.

Although tree-based representations have the advantages of reduced configuration space and fast packing time, we still should notice their limitations. For instance, the relationship between any two modules is unclear before the packing process, which may hinder the generation of an efficient placement solution, especially in the analog layout design. In TCG representation the transparent geometric relationship of modules is highly helpful for handling analog placement with complex constraints. Moreover, TCG potentialities of handling device merging or separation would benefit analog layout design with advanced constraints.

In the following chapter, we will introduce our proposed SP-Driven TCG method. We will also show how this approach can effectively and efficiently deal with analog placement.

3. Placement with SP-Driven TCG for Advanced Analog Constraints

In this chapter, we propose a sequence-pair (SP) driven transitive-closure-graph (TCG) method to effectively and efficiently deal with analog placement, which is an indispensable stage in the analog IC layout design. Besides discussing the promising benefit of the proposed method in handling advanced analog constraints, this chapter is more focused on the algorithmic complexity reduction when satisfying symmetry constraints. We also present a redundancy control scheme among the representation states in order to generate high-performance analog placement with high computation efficiency. Our experimental results demonstrate the efficacy of our proposed method.

3.1. Advantages of SP-Driven TCG for Advanced Analog Constraints

We believe TCG by nature provides such a capability of tackling the advanced analog constraints, since the relationship between each two vertices in TCG is well defined and the weight of each edge between two vertices can be used to control proximity or merging situations of two corresponding modules. Therefore, we investigate the TCG representation to handle advanced analog constraints. To open a door to that niche, in this section we will focus on a better symmetry-aware placement solution on top of the previous works in the literature. In particular, we propose an *SP-driven TCG method (SP-TCG for short)* to facilitate TCG operations by taking advantage of the SP's unique closure feature.

Before going into the details of our proposed SP-TCG representation, let's have a quick look back at the relevant topological representations previously published in the literature, including SP, TCG, TCG-S, and B*-tree.

Sequence Pair (SP)

SP is an ordered pair of module permutation sequences, which can be denoted as α-sequence and β-sequence, respectively. We can use SP to encode a placement, and then decode it to obtain the topological relationship between every two modules. The basic rules introduced in [13] are as follows. Module m_1 is to the left (right) of module m_2 if m_1 appears before (after) m_2 in both α- and β-sequences. Module m_1 is above (below) module m_2 if m_1 appears before (after) m_2 in α- sequence and m_1 appears after (before) m_2 in β-sequence. As an example, Figure 6(a) depicts a placement with five modules. Figure 6(b) is the corresponding SP of the placement in Figure 6(a). The α-sequence is (M_2, M_3, M_1, M_5, M_4), while the β-sequence is (M_1, M_2, M_3, M_4, M_5). As M_1 appears before M_5 in both α-sequence and β-sequence, M_1 is to the left of M_5. As M_4 appears after M_5 in α-sequence and before M_5 in β-sequence, M_4 is below M_5.

Transitive Closure Graph (TCG)

A TCG representation consists of two graphs, namely horizontal transitive closure graph H_{tcg} and vertical transitive closure graph V_{tcg}. Both graphs help describe the geometric relations among modules. In H_{tcg} (V_{tcg}), a vertex v represents a module m. The

value associated with a vertex indicates the width (height) of the corresponding module, and a directed edge $<v_i, v_j>$ represents that module m_i is to the left of (below) module m_j. For instance, Figure 6(c) shows the corresponding TCG representation of the placement in Figure 6(a). There is an edge between M_1 and M_5 in H_{tcg} and there is an edge between M_4 and M_5 in V_{tcg}.

Transitive Closure Graph-Sequence (TCG-S)

TCG-S representation combines TCG and part of SP representation. TCG-S is composed of a horizontal transitive closure graph H_{tcg}, a vertical transitive closure graph V_{tcg}, and the packing sequence, i.e., β-sequence of SP, which are to represent a placement. The β-sequence is the topological order of both H_{tcg} and V_{tcg}. Therefore, it doesn't need to execute extra operations on a TCG to extract the module packing sequence. Figure 6(d) is an example of TCG-S representation of the placement in Figure 6(a).

B*-tree

A B*-tree is a representation based on ordered binary-trees. The root of a B*-tree indicates the bottom-left comer module in a placement. Two modules represented by a node and its left child must have a left-right relationship. A node and its right child indicate that their corresponding modules below and above are related with each other. The left subtree of a node corresponds a set of modules, which are on the right side of the module represented by the node. Similarly, the right subtree of a node means the modules are

located on the upper side of the module, which corresponds to the node. An example of B*-tree is shown in the Figure 6(e), which corresponds the placement in Figure 6(a). For instance, N_4 is on the left subtree of N_1, while N_2 is on the right subtree of N_1.

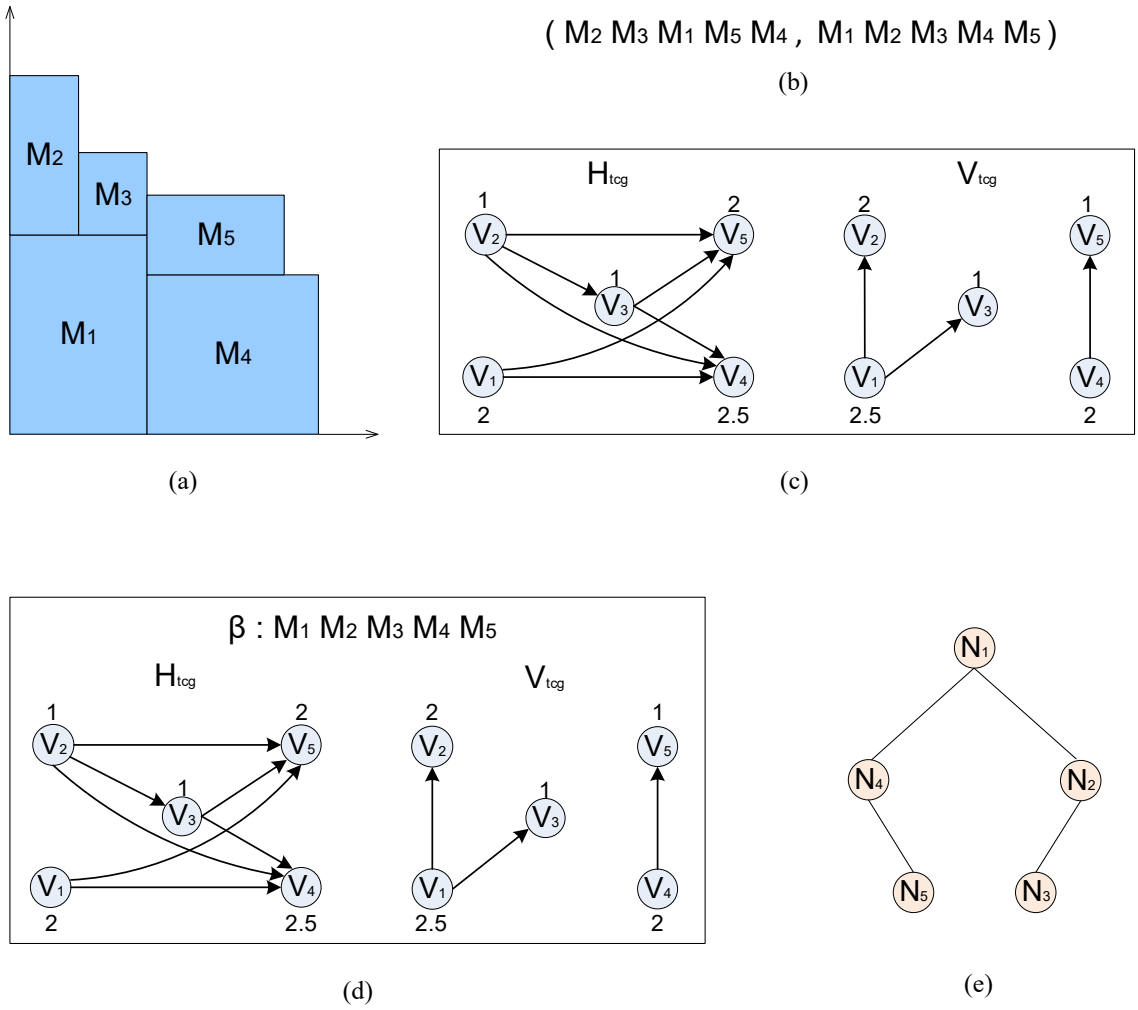


Figure 6 Relevant topological representations examples: (a) Corresponding placement, (b) SP, (c) TCG, (d) TCG-S, (e) B*-tree.

In Table I, we summarize the features of different placement approaches with topological representations, including SP, TCG, TCG-S, B*-tree, and this work. The last column "Advanced Constr." means whether the advanced analog constraints can be standalone explored by each individual approach.

Table I. Time complexity comparison of different topological representations in the context of symmetry and advanced constraints.

Approaches		*Packing*	*Perturbation*	*Advanced Constr.*
SP	General	*$O(n^2)$* [13]	*O(1)*	*No*
	Symmetry	*$O(n^2)$* [40]	*O(1)*	*No*
TCG	General	*$O(n^2)$* [18]	*$O(n^2)$*	*Yes*
	Symmetry	*$O(n^2)$* [44]	*O(n)*	*Yes*
TCG-S	General	*O(nlgn)* [19]	*O(n)*	*Yes*
	Symmetry	*$O(n^2)$* [46]	*$O(n^2)$*	*Yes*
B*-Tree	General	*O(n)* [16]	*O(n)*	*No*
	Symmetry	*O(n)* [20][21]	*O(lgn)*	*No*
SP-TCG	General.	*O(nlgn)*	*O(n)*	*Yes (This Work)*
	Symmetry	*O(nlgn)*	*O(n)*	*Yes (This Work)*

3.2. Symmetric-Feasible TCG and Placement

3.2.1 Symmetric-Feasible Conditions

In this work, we adopt the idea of symmetry island (called *cluster* throughout the chapter below) from [20], which is extended to the TCG representation. A symmetry cluster includes any self-symmetric modules and symmetric pairs belonging to one single symmetry group. Here, we use s_i and p_i to denote the i^{th} self-symmetric module and the right one of the i^{th} symmetric-pair modules, respectively. In the following, we define three symmetric-feasible conditions for an SP-driven TCG satisfying symmetry constraints.

Condition-1: s_i and s_j ($i \neq j$) must be straight vertically related.

Condition-2: s_i and p_j are horizontally or vertically related. Meantime, s_i can only be set on the left of p_j on the horizontal axis.

Condition-3: p_i and p_j ($i \neq j$) are horizontally or vertically related.

3.2.2 Symmetry-Aware Placement

Without loss of generality, here we assume the symmetry axis is vertical. We can use the right half of a self-symmetric module to represent itself. For a pair of symmetric modules, the right one is utilized as the representative. In this way, we can build a TCG from a placement consisting of the half part of a symmetric cluster. By following the convention of previous analog placement algorithms in the literature, we also use SA in

this work to optimize and derive the best placement solution. During the SA-based perturbation, the defined TCG symmetric-feasible conditions above should be always obeyed. Afterwards we can derive the left part to form a complete placement by mirroring the representative, i.e., the right part of the symmetric cluster.

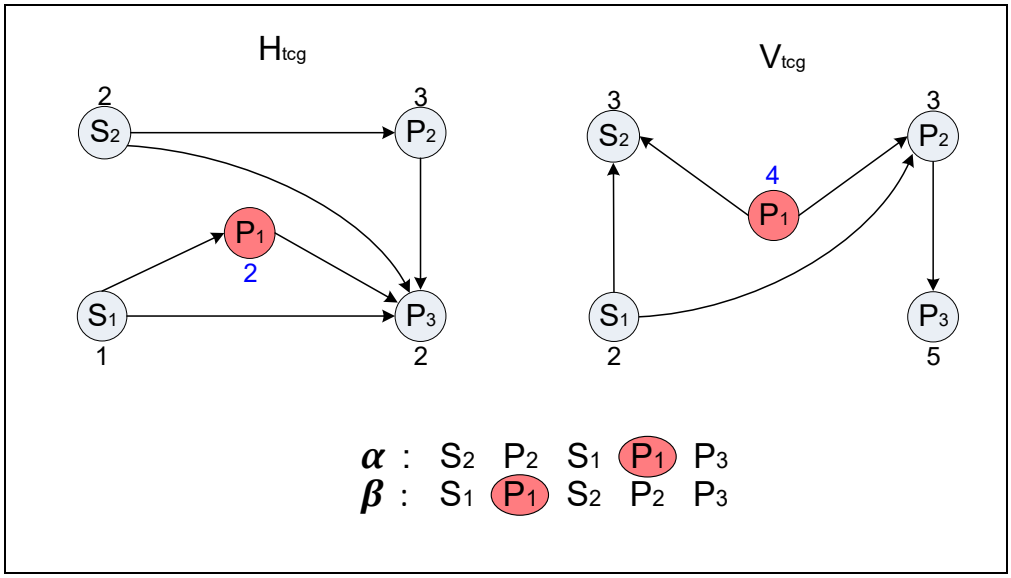
With the geometric outline of the complete placement, a super module, which is defined as the integral symmetric cluster, can be included into the original TCG representation. Putting it simply, our basic idea is to first pack a symmetric cluster into a super module for each symmetry group. Then asymmetric modules can be packed with all new generated super modules to form the final placement. For each packing operation, we can always use the same packing scheme in *O(nlgn)* time as introduced in [46], where *n* is the number of the modules.

3.3. SP-Driven Perturbation of Symmetric-Feasible TCG

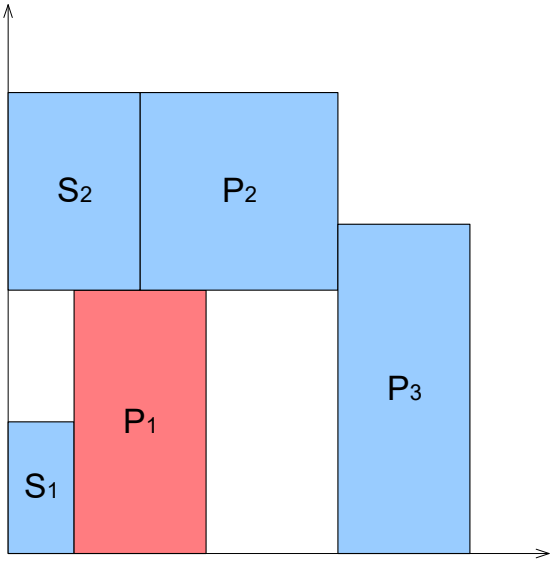
There is an intrinsic equivalent relationship between SP and TCG [19]. Therefore, we can utilize SP to guide the perturbation operations of a symmetric-feasible TCG. Our symmetric-feasible conditions and inherent properties of SP guarantee symmetric feasibility and validity of TCG after the random perturbation process.

3.3.1. Rotation

Orientation of any module or super module in a symmetric-feasible TCG and any symmetric-pair module in a super module is feasible to be tuned. We define such a geometric orientation change operation as *rotation*. Since a rotation operation does not change the topological structure of all the modules as a whole, it only needs constant time to rotate a non-self-symmetric module or a super module. Figures 7(a) and 8(a) show that we can rotate the module *P1* by swapping the corresponding nodes weights, which are *2* and *4* colored sky blue, in both H_{tcg} and V_{tcg}, while the sequence pairs keep the same as before. The resulted placement is depicted in Figure 8(b).

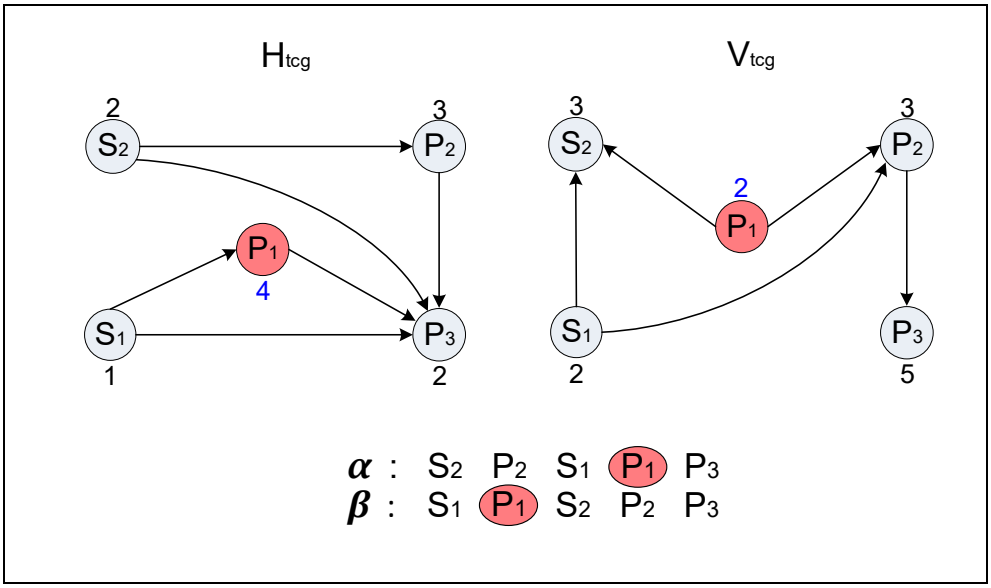


(a)

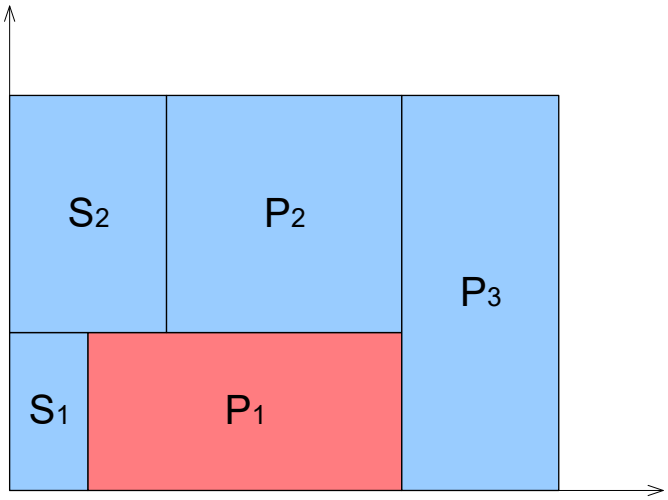


(b)

Figure 7 One example before the rotation operation: (a) SP-TCG, (b) Corresponding placement.



(a)



(b)

Figure 8 One example after the rotation operation: (a) SP-TCG, (b) Corresponding placement.

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

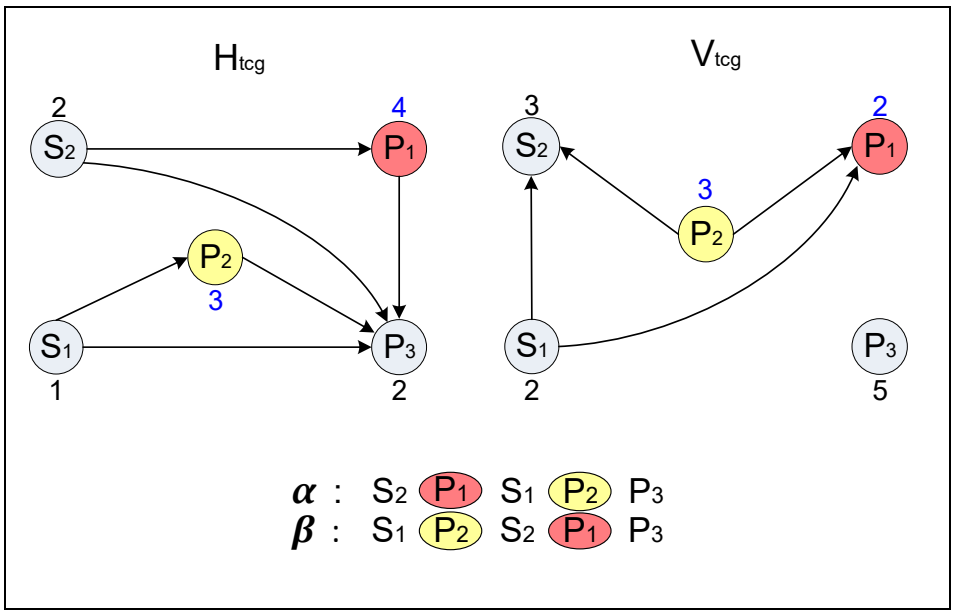
Proof: Due to the unchanged TCG topology after a rotation operation, the newly obtained TCG is still symmetric-feasible and valid. It only takes *O(1)* time to swap the weights of two nodes, which represent the same rotated non-self-symmetric module in both H_{tcg} and V_{tcg}. Q.E.D.

3.3.2. Exchange

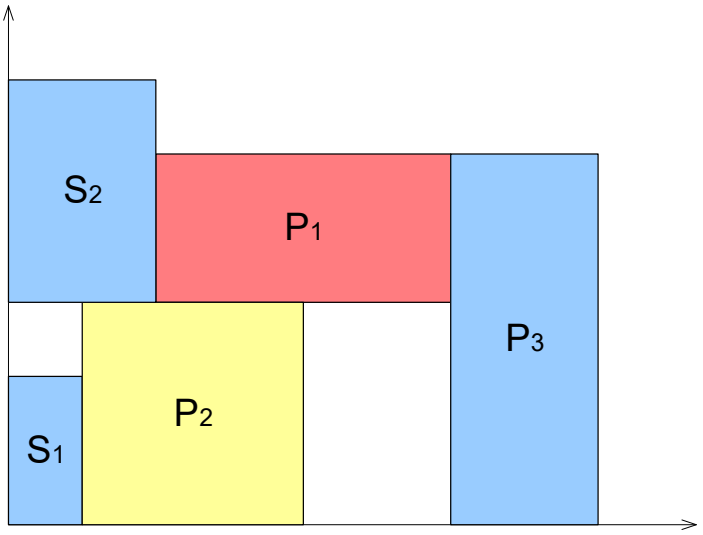
Any two modules including super modules in a symmetric-feasible TCG, two self-symmetric modules or two symmetric-pair modules in a super module are free to swap their geometric positions. Such an operation is defined as *exchange*. We can exchange any two modules in a symmetric-feasible TCG except for the following situation, which is a self-symmetric module and a symmetric-pair module are horizontally related to each other in a super module. If they are allowed to be exchanged, the symmetric feasible Condition-2 will be violated.

The example illustrated in Figures 9 and 10 presents how an exchange operation works. When we swap the nodes *P1* and *P2* along with their weights in both H_{tcg} and V_{tcg} as shown in Figures 9(a) and 10(a), the positions of the corresponding modules are exchanged. In Figure 9(b) module *P1* is on the top of module *P2*, while after the nodes

swapping module *P1* is changed to the bottom of module *P2*, whose result is depicted in Figure 10(b). In the meantime, the positions of *P1* and *P2* in the sequence pairs are swapped as well.

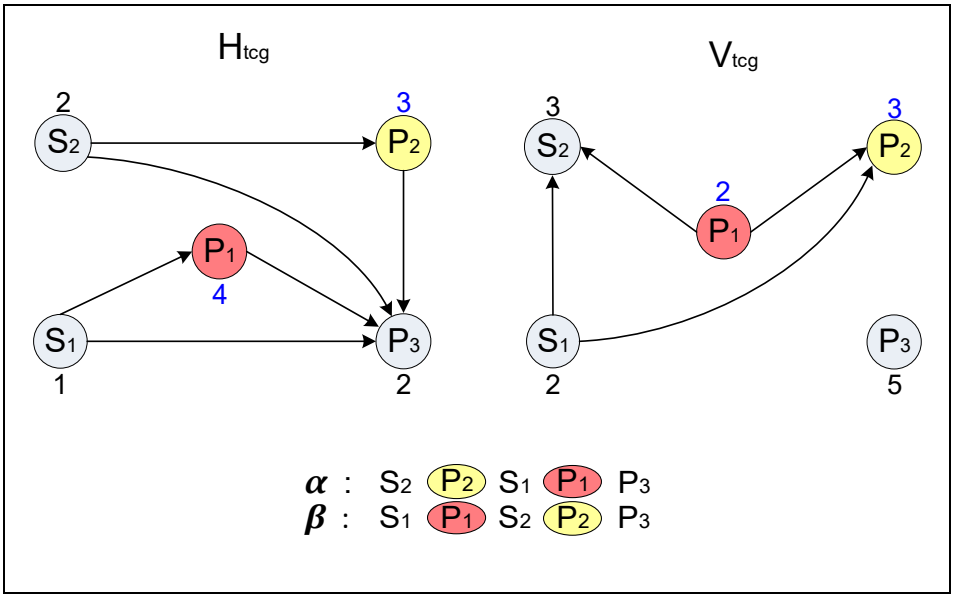


(a)

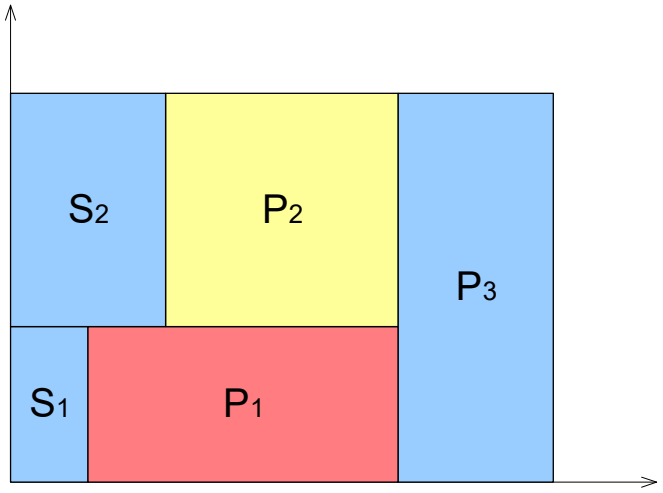


(b)

Figure 9. One example before the exchange operation: (a) SP-TCG, (b) Corresponding placement.



(a)



(b)

Figure 10. One example after the exchange operation: (a) SP-TCG, (b) Corresponding placement.

Lemma 2. Given a symmetric-feasible TCG, the resulting TCG after an exchange operation is still symmetric-feasible and valid, and it takes *O(1)* time.

Proof: Since the exchange operation only swaps two nodes in both constraint graphs without changing the topology of the original TCG, the resulting graphs are still symmetric-feasible and valid. Exchanging the corresponding two nodes only takes *O(1)* time. Q.E.D.

3.3.3 Change

Choosing any two modules in α-sequence, then moving the edges between the corresponding nodes in a valid TCG (either horizontal or vertical) to its counterpart graph, is defined as *SP-driven move*. Similarly, picking any two modules in β-sequence, then moving the edges between the corresponding nodes in a valid TCG (either horizontal or vertical) to its counterpart graph, and changing their direction is defined as *SP-driven move reverse*. When we utilize the above-defined operations to a symmetric-feasible TCG and eventually still satisfy the symmetric-feasible conditions, both operations are called *SP-driven symmetric move* and *SP-driven symmetric move reverse*, respectively. We define a series of SP-driven move, SP-driven move reverse, SP-driven symmetric move, and SP-driven symmetric move reverse operations as *SP-driven change* operation.

We use SP-driven change operation to vary the relationship between two modules in a symmetric-feasible TCG. The perfect sequence closure nature of SP facilitates and

guarantees the validity of the varied TCG through a series of SP-guided move and/or move reverse operations, while it does likewise to a symmetric-feasible TCG with the aid of the symmetric-feasible conditions. The following examples help describe the details of the SP-driven change operation with its pseudocode presented at first in a general way.

SP-driven move operation

Algorithm: SPdrivenMove

Input: α-sequence, β-sequence and a valid TCG

Output: a valid updated TCG after α-sequence-driven move

Begin

1 randomly choose two different modules *a* and *b*

2 save the modules between *a* and *b*, exclusive of *a*, in the α-sequence into set *M*

3 **for** (each module *c* in set *M*)

4 check the relationship between modules *a* and *c*

5 move the edge between *a* and *c* to the counterpart graph

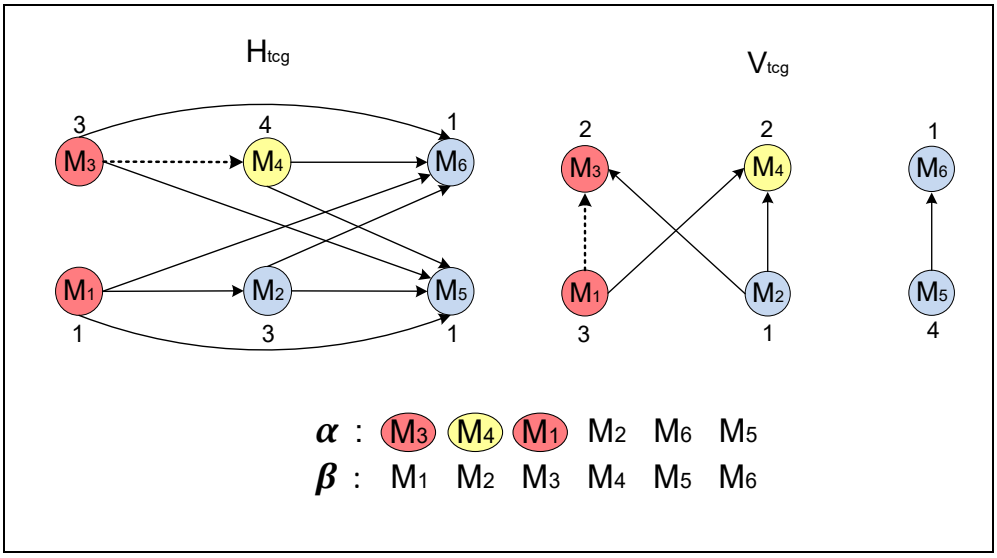
6 **endfor**

7 update α-sequence

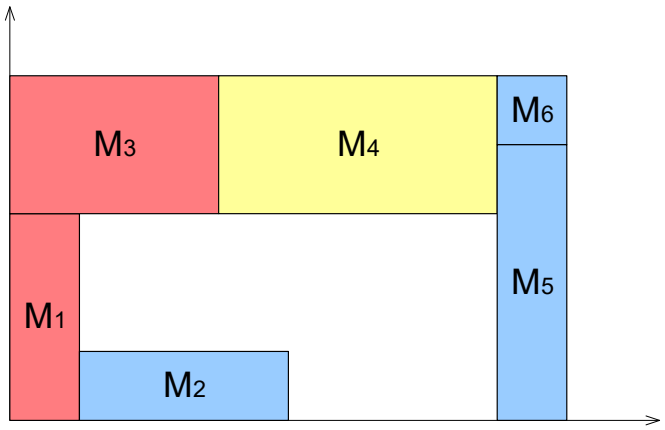
End

Figure 11. Pseudocode of the SP-driven move operation.

The pseudocode of the SP-driven move operation is listed in Figure 11. Here we use an example illustrated in Figures 12 and 13 to show how this operation works efficiently. Modules M3 and M1 in Figure 12 are randomly chosen in α-sequence, which leads to changing the relationship between M3 and M4, then between M3 and M1, as shown by the dashed edges. The dashed edge between M3 and M4 is moved from H_{tcg} as shown in Figure 12(a) to V_{tcg} as depicted in Figure 13(a), while the dashed edge between M3 and M1 is transferred from V_{tcg} in Figure 12(a) to H_{tcg} in Figure 13(a). The obtained placement in Figure 13(b) shows the effectiveness of the SP-driven move operation. Since there is only one for-loop, the analysis of our algorithm *SPdrivenMove* in Figure 11 shows its time complexity is $O(n)$, where n is the number of modules.

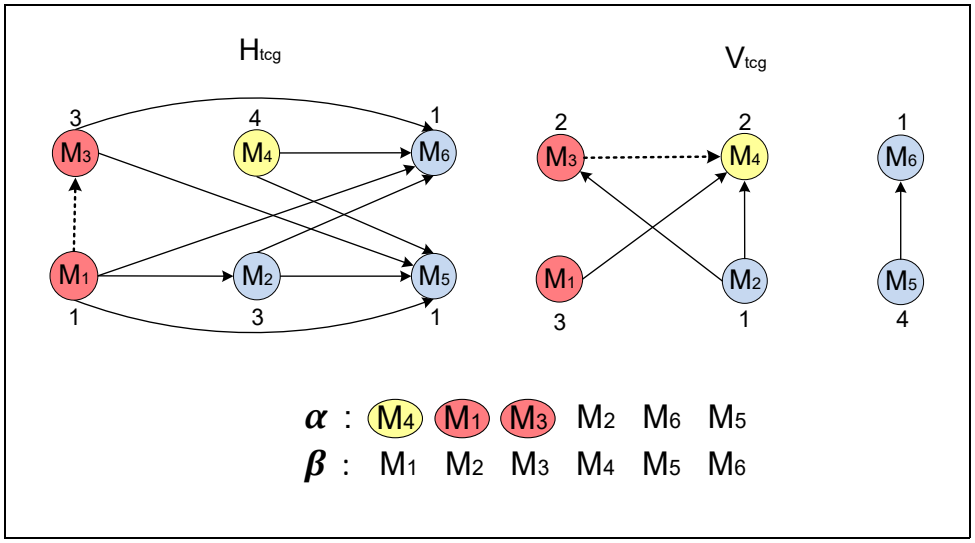


(a)

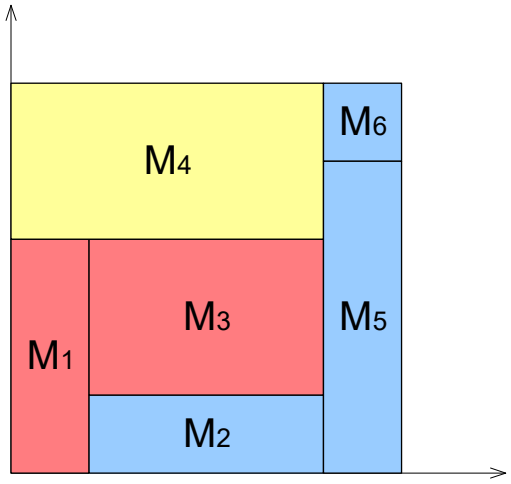


(b)

Figure 12. One example before the SP-driven move operation: (a) SP-TCG, (b) Corresponding placement.



(a)



(b)

Figure 13. One example after the SP-driven move operation: (a) SP-TCG, (b) Corresponding placement.

SP-driven move reverse operation

The SP-driven move reverse operation works in a similar way, while two modules are randomly chosen in β-sequence other than α-sequence. If we describe the operations in terms of the graphs, we will not only move the edge between these two selected modules to the counterpart graph, but also flip over the edge. The pseudocode of the SP-driven move reverse operation is listed in Figure 14.

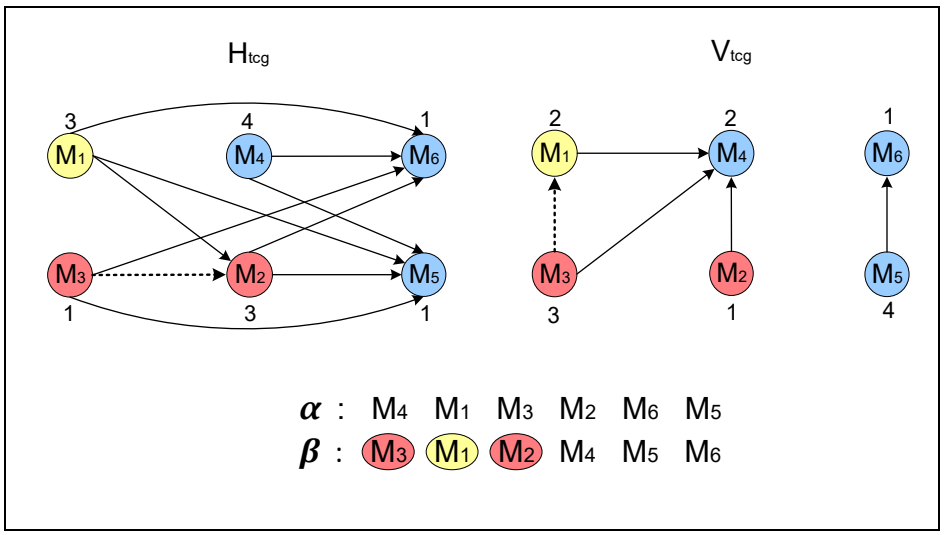
Algorithm: SPdrivenMoveReverse

Input: α-sequence, β-sequence and a valid TCG
Output: a valid updated TCG after β-sequence-driven move reverse

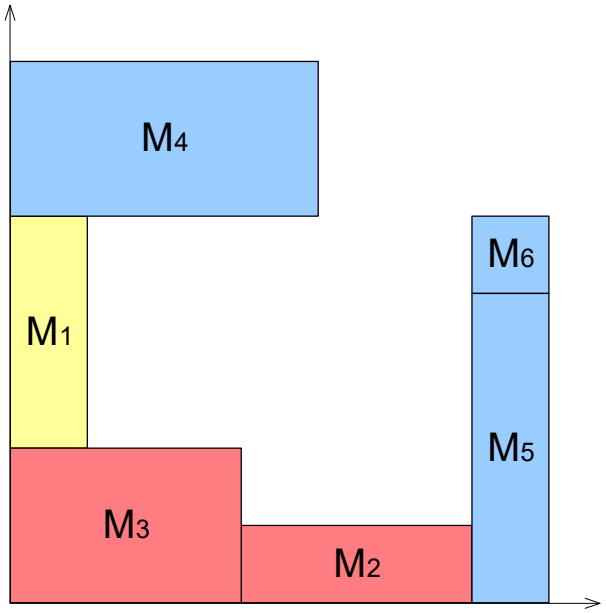
Begin
1 randomly choose two different modules *a* and *b*
2 save the modules between *a* and *b*, exclusive of *a*, in the β-sequence into set *M*
3 **for** (each module *c* in set *M*)
4 check the relationship between modules *a* and *c*
5 move and reverse the edge between *a* and *c* to the counterpart graph
6 **endfor**
7 update β-sequence
End

Figure 14. Pseudocode of the SP-driven move reverse operation.

The example illustrated in Figures 15 and 16 shows the high efficiency of the SP-driven move reverse operation. We randomly choose modules *M3* and *M2* in β-sequence as shown in Figure 15. The dashed edges are used to present the change of the relationship between *M3* and *M1*, and then between *M3* and *M2*. While the dashed edge between *M3* and *M1* is moved and reversed from V_{tcg} as depicted in Figure 15(a) to H_{tcg} in Figure 16(a), the converse dashed edge between *M3* and *M2* is transferred from H_{tcg} in Figure 15(a) to V_{tcg} in Figure 16(a). The obtained placement in Figure 16(b) shows the effectiveness of the SP-driven move reverse operation. Since there is only one for-loop, the analysis of our algorithm *SPdrivenMoveReverse* in Figure 14 shows its time complexity is *O(n)*, where *n* is the number of modules.

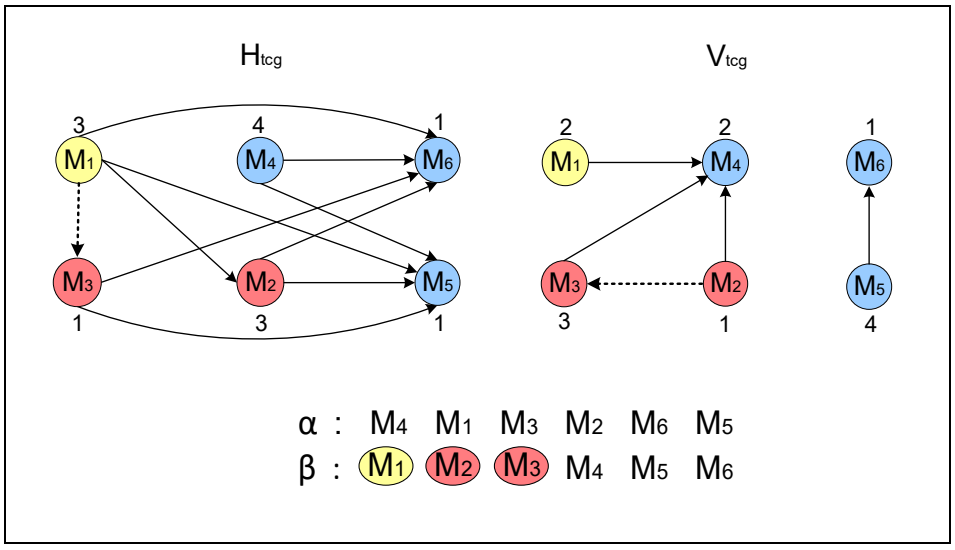


(a)

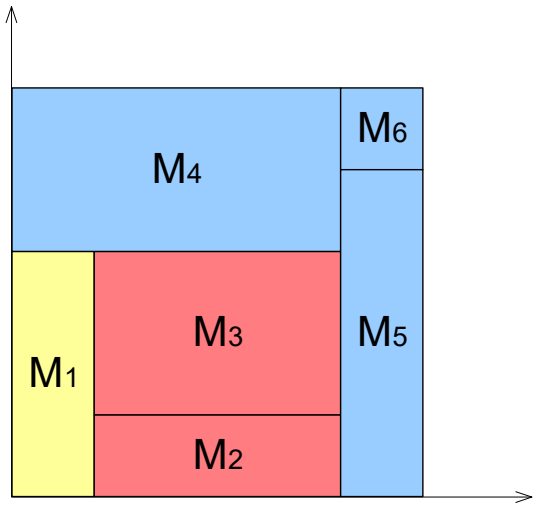


(b)

Figure 15. One example before the SP-driven move reverse operation: (a) SP-TCG, (b) Corresponding placement.



(a)



(b)

Figure 16. One example after the SP-driven move reverse operation: (a) SP-TCG, (b) Corresponding placement.

SP-driven symmetric move operation

We apply the SP-driven symmetric move operations to perturb symmetric-feasible TCG, which represents the right half of a symmetry cluster. The pseudocode of the SP-driven symmetric move operation is listed in Figure 17. It is similar to the general case, and its time complexity keeps $O(n)$, where n is the number of symmetric modules.

Algorithm: SPdrivenSymMove

Input: α-sequence, β-sequence and a symmetric-feasible TCG

Output: an updated symmetric-feasible TCG after α-sequence-driven symmetric move operation

Begin

1 randomly choose a number between 0 and the total module number minus 1 as loop number N_l

2 **for** (each loop through N_l)

3 randomly choose two different modules a and b, b is next to a *on the right*

4 **if** (a is a symmetric-pair module)

5 check the relationship of a and b

6 move the edge between a and b to the counterpart graph

7 **else**

8 **if** (a is a self-symmetric module && b is a symmetric-pair module && a is on the left of b in β-sequence)

9 check the relationship of a and b

10 move edge between a and b to the counterpart graph

11 **else**

12 continue

13 **endif**

14 **endif**

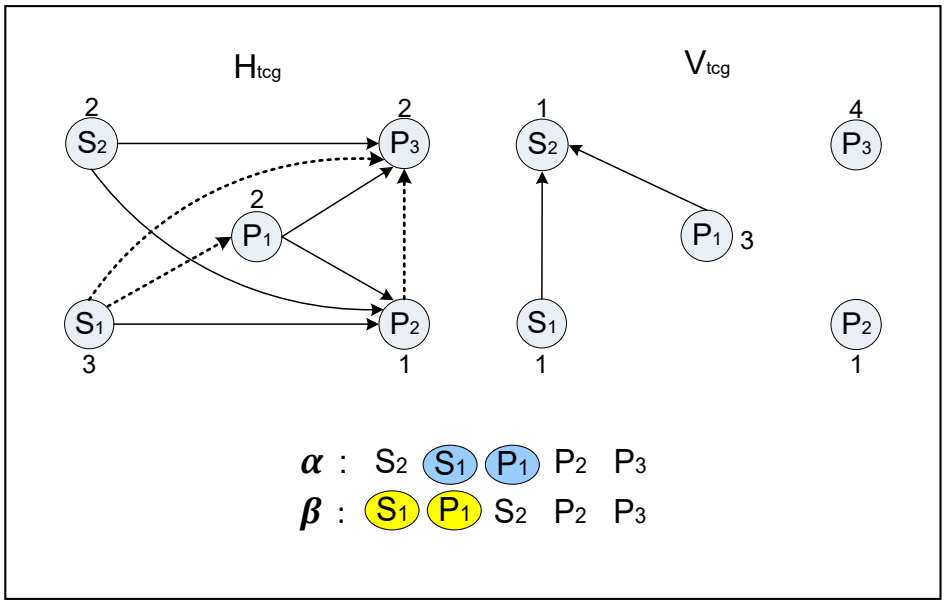
15 update α-sequence

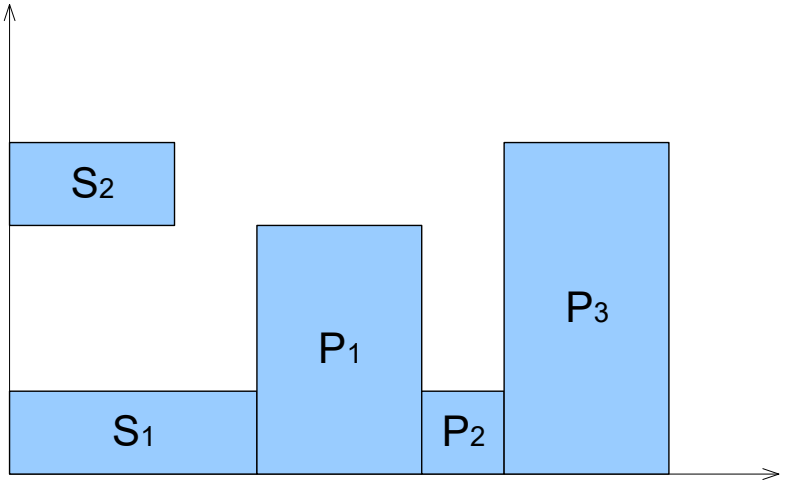
16 **endfor**

End

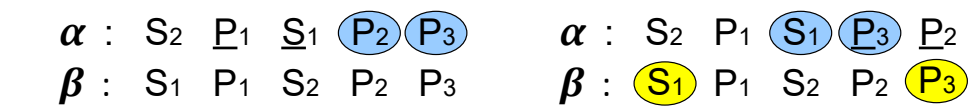
Figure 17. Pseudocode of the SP-driven symmetric move.

Let us use another example to illustrate SP-driven symmetric move method. Figure 18 illustrates how to change a symmetric-feasible TCG through three consecutive SP-driven symmetric move operations.

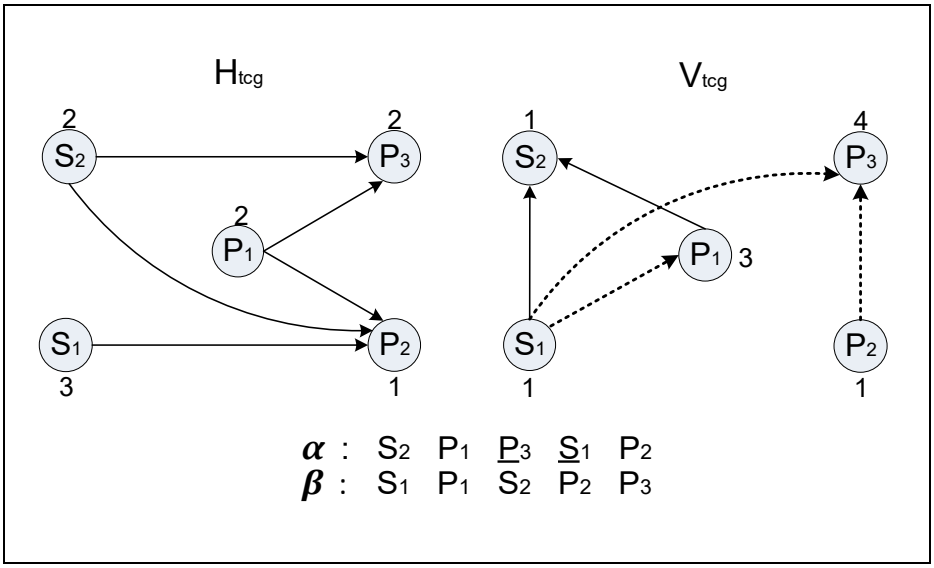


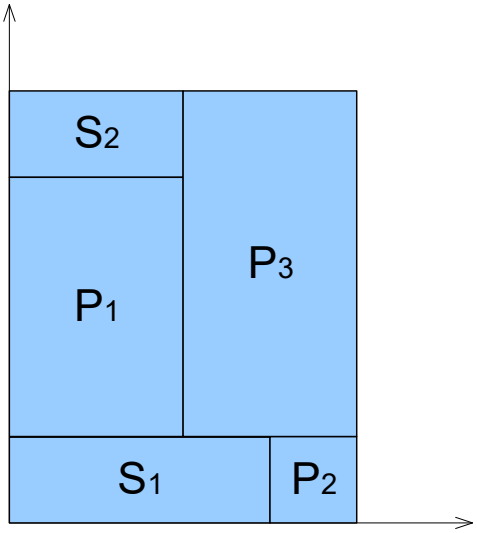


(a)



(b)





(c)

Figure 18. One example of three consecutive SP-driven symmetric move operations: (a) SP-TCG and its corresponding placement before the operations, (b) SP during the operations, (c) SP-TCG and its corresponding placement after the operations.

Figure 18(a) depicts that *S1* and *P1*, as symmetric representatives, are randomly selected from α-sequence and then the dashed edge between *S1* and *P1* is moved from H_{tcg} to V_{tcg}. Here before the move, we have to check the symmetry type of *S1* and *P1*. Since self-symmetric *S1* is on the left of symmetric-pair *P1* in both α- and β-sequences (i.e., *S1* is topologically located on the left of *P1*), the move operation mentioned above (i.e., *P1* is moved to the top of *S1*) is acceptable. Otherwise, such a move operation is not allowed.

As the next operation, *P2* and *P3* are picked at random in α-sequence as shown in Figure 18(b). Since *P2* is a symmetric-pair module, no other conditions are required to be satisfied. So we can directly move the dashed edge between *P2* and *P3* from H_{tcg} to V_{tcg}, as illustrated in Figures 18(a) and (c). The third move operation is similar to the first one as shown in Figure 18(c), where one can clearly see the perturbation effectiveness in terms of compactness of the final placement.

SP-driven symmetric move reverse operation

The SP-driven symmetric move reverse operation functions like the SP-driven symmetric move operation, but on the β-sequence. When we move the edge between two modules to the counterpart graph, we also need to reverse the edge to its opposite direction. The symmetric move reverse operations are also used to perturb symmetric-feasible TCG. Figure 19 gives the pseudocode of the SP-driven symmetric move reverse operation. Similarly to the general case, its time complexity remains $O(n)$, where n is the number of symmetric modules.

Algorithm: SPdrivenSymMoveReverse

Input: α-sequence, β-sequence and a symmetric-feasible TCG

Output: an updated symmetric-feasible TCG after β-sequence-driven symmetric move reverse operation

Begin

1 randomly choose a number between 0 and the total module number minus 1 as loop number N_l

2 **for** (each loop through N_l)

3 randomly choose two different modules a and b, b is next to a *on the right*

4 **if** (a is a symmetric-pair module)

5 check the relationship of a and b

6 move and reverse the edge between a and b to the counterpart graph

7 **else**

8 **if** (a is a self-symmetric module && b is a symmetric-pair module && a is on the left of b in α-sequence)

9 check the relationship of a and b

10 move and reverse the edge between a and b to the counterpart graph

11 **else**

12 continue

13 **endif**

14 **endif**

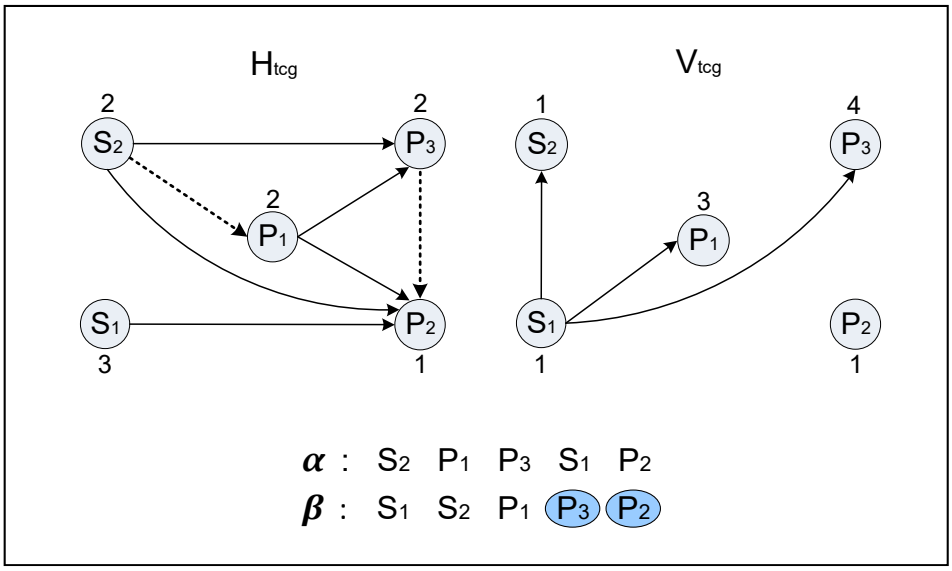
15 update β-sequence

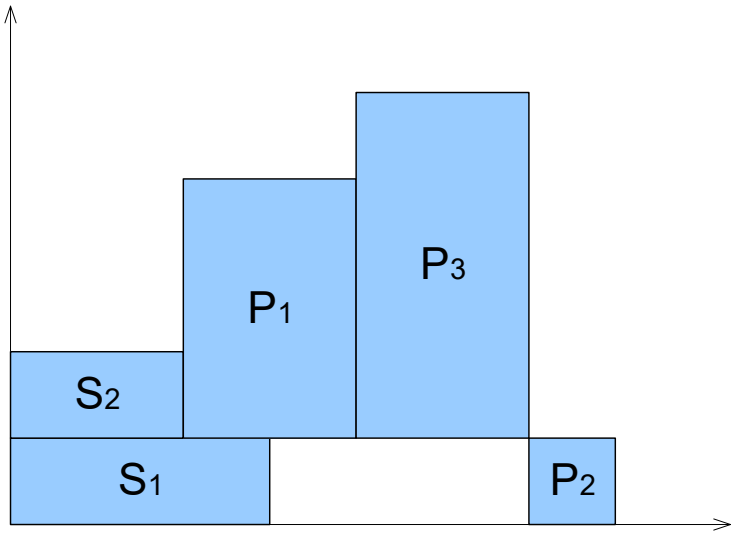
16 **endfor**

End

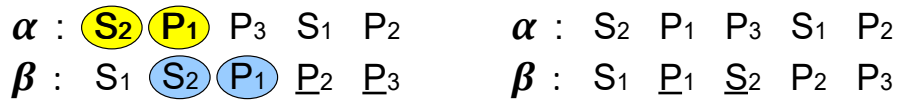
Figure 19. Pseudocode of the SP-driven symmetric move reverse.

The example depicted in Figure 20 illustrates the mechanism of the SP-driven symmetric move reverse operation. It presents that we can use two (or more) consecutive SP-driven symmetric move reverse operations to effectively change a symmetric-feasible TCG.

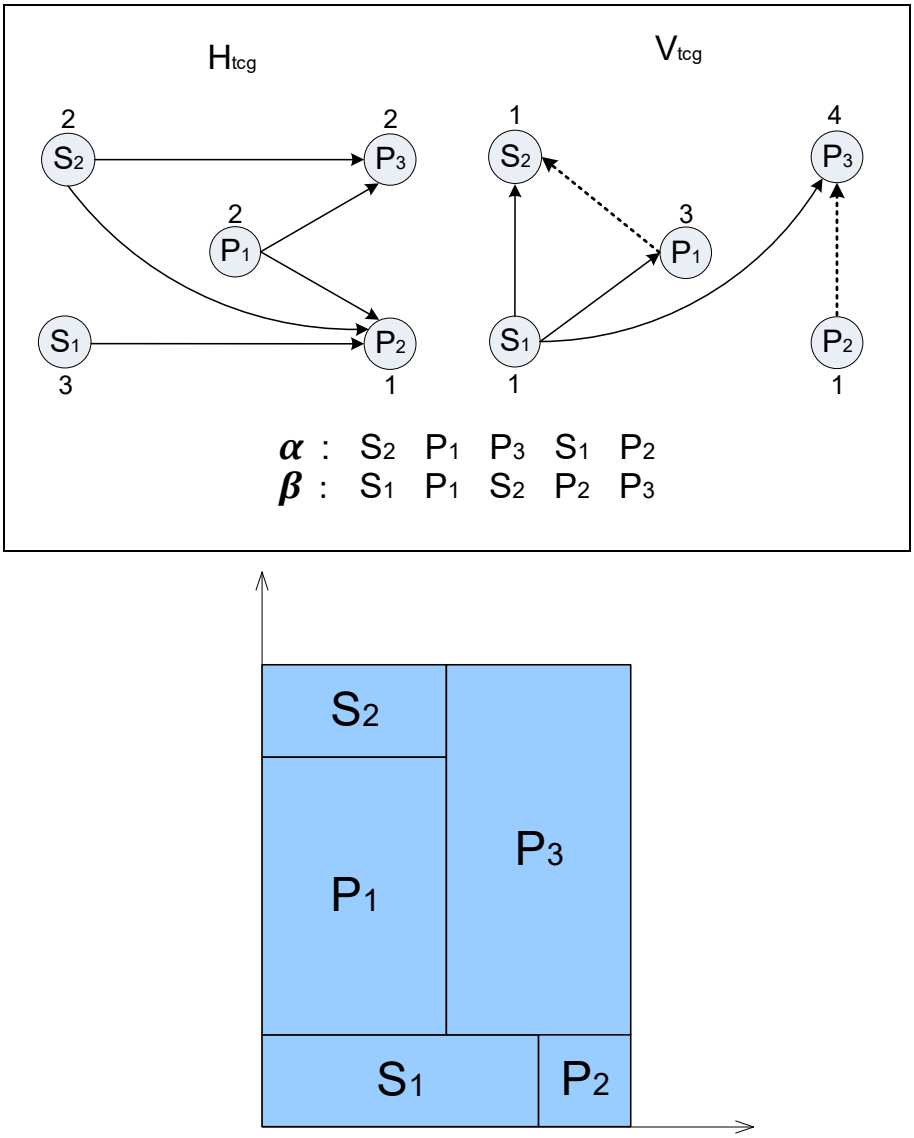




(a)



(b)



(c)

Figure 20. One example of two consecutive SP-driven symmetric move reverse operations: (a) SP-TCG and its corresponding placement before the operations, (b) SP during the operations, (c) SP-TCG and its corresponding placement after the operations.

Figure 20(a) shows an initial SP-driven TCG with the corresponding placement. The SP changes during the move reverse operations are indicated in Figure 20(b). We first randomly select *P3* and *P2*, as symmetric representatives, from β-sequence. Since *P3* is a symmetric-pair module, we can directly move and reverse the dashed edge between *P3* and *P2* from H_{tcg} to V_{tcg}, as illustrated in Figures 20(a) and (c).

Then *S2* and *P1* are picked at random from β-sequence, while their positions in α-sequence are needed to be checked. Because self-symmetric *S2* is on the left of symmetric-pair *P1* in both α- and β-sequences (i.e., *S2* is topologically located on the left of *P1*), we can move and reverse the dashed edge between *S2* and *P1* from H_{tcg} to V_{tcg} (i.e., *P1* is moved to the bottom of *S2*), as depicted in Figures 20(a) and (c). Figure 20(c) exhibits the compact final placement result after the effective perturbations as listed above.

Lemma 3. Given a symmetric-feasible TCG, an SP-driven change operation guarantees symmetric feasibility and validity of the resulting TCG. Such an operation takes *O(n)* time, where *n* is the number of modules.

Proof: Given two symmetric modules within a symmetric cluster, there are total four types of combination: $m_s m_s$, $m_p m_p$, $m_s m_p$, and $m_p m_s$, where m_s stands for any self-symmetric module and m_p represents any representative symmetric-pair module. Below we will analyze each situation in detail.

(1) $\boldsymbol{m_s m_s}$: The relationship of two distinct self-symmetric modules can only be represented in SP as ($m_{S2} m_{S1}$) ($m_{S1} m_{S2}$) or ($m_{S1} m_{S2}$) ($m_{S2} m_{S1}$). If we change any one of these

sequences, for instance, ($m_{S1}m_{S2}$) ($m_{S1}m_{S2}$), the relation of m_{S1} and m_{S2} is turned from vertical to horizontal, which contradicts symmetric-feasible Condition-1.

(2) $\boldsymbol{m_pm_p}$: SP ($m_{p1}m_{p2}$) ($m_{p1}m_{p2}$) or ($m_{p2}m_{p1}$) ($m_{p2}m_{p1}$) represents that two distinct representative symmetric-pair modules are horizontally related. We can tell that any change on α or β-sequence, for example, ($m_{p2}m_{p1}$) ($m_{p1}m_{p2}$), makes the relationship of two modules turn to vertical. Meanwhile, symmetric-feasible Condition-3 is still satisfied. This is also true for the vertical relationship originally between m_{p1} and m_{p2}.

(3) $\boldsymbol{m_sm_p}$ **or** $\boldsymbol{m_pm_s}$: When m_s is horizontally located on the left of m_p, SP (m_sm_p) (m_sm_p) indicates that change of any one sequence leads to a vertical relationship of these two modules. However, the state represented by SP (m_pm_s) (m_pm_s) violates symmetric-feasible Condition-2.

Based on the analysis above, all these situations correspond to two main conditions as Line-4 and Line-8 in Figure 19. Moreover, every SP can generate a feasible placement and correspond to an equivalent valid TCG. Therefore, an SP-driven change can assure both symmetric feasibility and validity of the resulting TCG. In Figure 19, Line-1 implies that there are at most n-1 modules in set M, n is the total number of modules. Thus, the worst case is n-1 modules need to be checked. As for the identification of relationship between two modules, the edge move operation between the graphs, and the update of sequences, all the relevant operations only take $O(1)$ time. Hence, the SP-driven move operation takes $O(n)$ time, where n is the number of modules. Similarly, the SP-driven move reverse operation also needs $O(n)$ time. In contrast, the symmetric verification only requires $O(1)$ time. The SP-driven symmetric move reverse operation is alike, taking $O(n)$

time as well. Therefore, the time complexity of an SP-driven change operation is *O(n)* time, where *n* is the number of modules. Q.E.D.

Theorem 1. The solution space of symmetric-feasible SP-driven TCG can be fully explored by using random rotation, exchange, and change operations. The transformation of two TCG configurations takes at most *O(n)* time, where *n* is the number of modules.

Proof: We may clearly view the exploration of the TCG configuration space with the aid of the corresponding SP. According to the features of the rotation, exchange and change operations, a module can be randomly changed to any valid position in both α- and β-sequences. In addition, according to Section 3.2.2, it is ensured that the perturbed TCG satisfying the symmetric-feasibility conditions can map to a valid symmetric placement. This means the full exploration of the solution space can be performed by these operations. Obviously, the time complexity of the operations above is *O(n)*, where *n* is the number of modules. Q.E.D.

3.4 Perturbation Redundancy Control

During SP-driven TCG perturbation, some states may repeatedly occur due to randomness. We anticipate that redundant packing process would waste considerable time. Therefore, we develop a tactic, called *perturbation redundancy control*, to prevent such unnecessary repetition. In this regard, we establish a data base to hold distinct SP-TCG

states. After each perturbation, we compare the resultant SP-TCG states with those in the data base. If they are the same, we will reuse the recorded packing result and then go to the next perturbation. Otherwise, the new state will be packed and recorded into the data base along with the packing result. This approach can ensure that all recorded states are different.

If we check a TCG represented state, we have to deal with all the information contained in the graphs, such as nodes, edges between every two nodes, edge directions and weights. However, with the aid of SP that is equivalent to TCG per se, we only need to compare two sequences besides module orientation information. Evidently, SP is more effective to the perturbation redundancy control, which can significantly improve the efficiency of our developed SP-driven symmetric-feasible TCG perturbation.

3.5 Experimental Results

We have developed an SA-based placement algorithm for analog layout designs by using the proposed SP-driven TCG method. According to *Theorem 1*, we can take advantage of the operations introduced in Section 3.3 to explore the solution space of valid symmetric placements. Compared with the general symmetry-free TCG [18] and symmetry-aware TCG [44] placement approaches, our symmetry-aware SP-driven TCG method can achieve the fastest solution perturbation in $O(n)$ time and the lowest packing time complexity, i.e., $O(nlgn)$, where n is the number of modules, as shown in Table I.

To test the performance of our proposed algorithm, it has been coded in C++ and applied to several test circuits on an Intel X86 1.2GHz Linux workstation that has 64GB of

memory. We first evaluated the effectiveness of our proposed perturbation redundancy control scheme. We implemented the representation of SP states by using C++ Standard Template Library (STL) vector and C++ string (considering α-sequence, β-sequence, and module orientation information). The comparison of these two with the plain control-free implementation (denoted as "*Plain*") is listed in Table II. It was found that the perturbation state redundancy widely existed in our experiments. For our three test circuits with various module numbers (within the range of common analog circuits), the redundancy rate varied from 6.7% to 50.8%. Due to the large overhead of C++ STL vector, its run time was even longer than the plain implementation. But the C++ string implementation could readily improve the runtime efficiency by at least 20% compared to the plain implementation.

Table II. Comparison of various perturbation redundancy control implementations.

Test Circuits		**Circuit-1**	**Circuit-2**	**Circuit-3**
#Modules		10	18	25
Plain	#States	3931	6170	10091
	Time (sec.)	0.55	2.55	6.31
Vector	#States	3569	5720	5397
	Time (sec.)	2.53	6.68	20.21
String	#States	3360	5753	4966
	Time (sec.)	0.44	1.73	4.28

To demonstrate the efficiency of our proposed SP-driven TCG method, two other approaches coded in C++ on the same platform have been included for comparison: (i)AbsPlace, one absolute placement scheme using absolute coordinates [7]; and (ii) HB*-trees, an implementation that imitates [20][21]. The comparison results are given in Table III, where less packing cost (denoted "*PackCost*") and less execution time (denoted "*T (sec.)*") are preferable. It can be seen that our proposed method SP-TCG outperforms the other two approaches in the search quality. On average, compared with HB*-trees, SP-TCG reduces the cost by 5.1% with slightly shorter CPU time. When compared with AbsPlace, SP-TCG reduces the cost by 23.7% and the execution time by 36.3% on average.

Table III. Comparison of alternative methods on three test circuits.

Test Circuits		*AbsPlace*	*HB*-Trees*	*SP-TCG*
Circuit-1	PackCost	6.63e+07	5.41e+07	5.33e+07
	T (sec.)	0.67	0.45	0.44
Circuit-2	PackCost	9.09e+05	7.08e+05	6.67e+05
	T (sec.)	2.72	1.89	1.73
Circuit-3	PackCost	1.43e+06	1.15e+06	1.05e+06
	T (sec.)	7.07	4.44	4.28

3.6 Summary

In this chapter, we presented a sequence-pair-driven TCG representation and its associated method to facilitate the handling of advanced analog placement constraints, with a focus on the implementation of symmetry-aware placement. We defined our special symmetric-feasible conditions and described the construction of symmetric placement with the SP-driven TCG representation. We also proposed SP-driven perturbation operations and redundancy control scheme. Our experimental results have shown that this proposed approach can generate high-performance placement with satisfactory computation efficiency.

In the next chapter, we will present our proposed ATCG placement representation for analog layout design. The SP-driven TCG representation, symmetric-feasible conditions, and associated operations developed in this chapter can be directly used in the ATCG-based placement method. The transformation between placement and ATCG will be deliberated and meanwhile the properties of ATCG will be proved. We will also demonstrate the effectiveness of our proposed representation.

4. Advanced TCG-Based Placement Representation for Analog Layout Design

In this chapter, we propose an advanced transitive-closure-graph-based placement representation (ATCG) to facilitate analog layout design. The versatility and flexibility of ATCG can ensure it to accurately control spacing and merging constraints uniquely required by analog layout design. Our experimental results demonstrate that our proposed representation can help generate high-performance analog placement with high computation efficiency.

4.1. ATCG Placement Representation

As presented in Chapter 3, TCG owns some features highly beneficial to the analog placement problem by nature. In this work, we have been exploiting the potentialities of TCG, including clear relationship between each two vertices and specific weight of each edge between two vertices, in order to perform better control on spacing and merging situations of two corresponding modules. In this regard, we propose to use ATCG as a powerful representation to handle advanced analog constraints.

In Table IV, we summarize the features of different placement approaches with topological representations, including SP, TCG, TCG-S, CBL, HB*-trees, QB-trees, and this work. The last two columns show whether the spacing and merging constraints can be explored by each individual representation.

Table IV. Comparison of different topological representations in the context of symmetry and advanced constraints.

Placement Representation	*Symmetry Packing*	*Symmetry Perturb*	*Merging Constraint*	*Spacing Constraint*
SP [41]	$>O(n^2)$	*O(1)*	*No*	*No*
TCG [44]	$O(n^2)$	*O(n)*	*No*	*No*
TCG-S [46]	$O(n^2)$	$O(n^2)$	*No*	*No*
CBL [49]	*O(n)*	*O(n)*	*No*	*No*
HB*-Trees [20]	*O(n)*	*O(lgn)*	*No*	*No*
QB-Trees [21]	*O(n)*	*O(lgn)*	*No*	*No*
ATCG (This Work)	*O(nlgn)*	*O(n)*	*Yes*	*Yes*

In this section, we first present how to construct horizontal and vertical constraint graphs for ATCG from a placement. Then we describe how to gain fast module packing from ATCG. Lastly, we discuss the properties of ATCG.

4.1.1. From a Placement to the Corresponding ATCG

An ATCG placement representation uses two transitive closure graphs, namely a horizontal constraint graph H_{atcg} and a vertical constraint graph V_{atcg}, to picture the topological relationships among modules. Both graphs are directed acyclic weighted graphs, which can be generally defined as $G = (V, E)$, where $E = \{(v_i, v_j), w_{ij} \mid v_i \in V, v_j \in V, w_{ij} \geq 0, i \neq j\}$. The edges, which are weighted, represent the minimum distance requirement of two modules, based on their bottom-left module corners. In H_{atcg}, we assume there are two vertices v_i and v_j, which represent two modules m_i and m_j respectively, and a directed edge from v_i to v_j. The edge weight w_{ij} can be set as w_i, which denotes width of m_i, if both modules abut closely for gaining compact placement. If m_i and m_j are required to be merged, we need to deduct their overlapping size from w_i (recorded as w_{ij}). If w_{ij} is greater than w_i, both modules are separated with a distance in between. The definition is similar for V_{atcg}, where we use modules height instead of width.

Figure 21 illustrates a valid ATCG, which is derived from a placement including six modules as shown in Figure 22. We denote a horizontal (vertical) relationship by $\rightarrow$ ($\uparrow$). If two modules have both a primary horizontal (vertical) relation and a secondary vertical (horizontal) relation, namely dual relation, we use notation $\Uparrow$ ($\Rightarrow$) to indicate such a secondary relation. These notations are used to describe the process of obtaining the corresponding ATCG from a placement below.

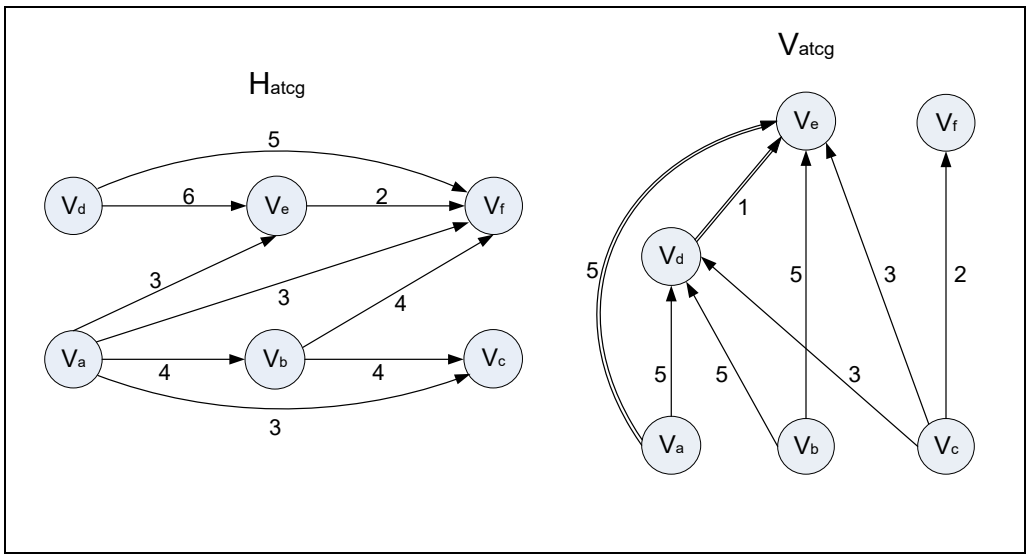


Figure 21. One example of a valid ATCG.

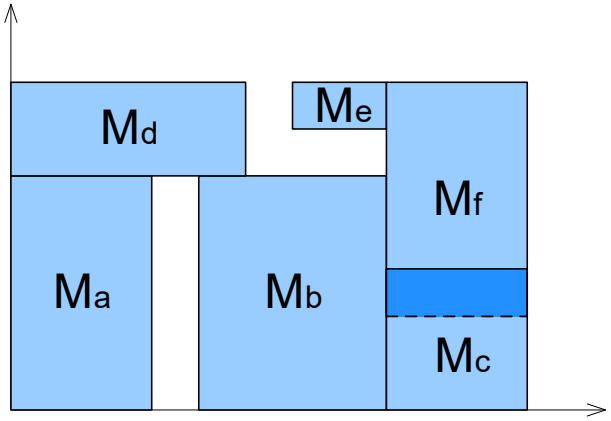


Figure 22. Corresponding placement of the valid ATCG.

We set vertex v_i in H_{atcg} and V_{atcg} respectively for each module m_i in a placement. If $m_i \rightarrow m_j$, we connect v_i to v_j with a directed weighted edge (v_i, v_j) in H_{atcg}. Similarly, we use a directed weighted edge (v_i, v_j) to link v_i to v_j in V_{atcg} if $m_i \uparrow m_j$. If $m_i \Rightarrow m_j$ or $m_i \Uparrow m_j$ exists, we may opt to set a directed weighted edge (v_i, v_j) to connect v_i with v_j in both H_{atcg} and

V_{atcg}. Thus, an ATCG is established after all the relations between every two modules are checked and reflected in H_{atcg} and V_{atcg}.

Figure 21 shows that six vertices v_a-v_f are placed in H_{atcg} and V_{atcg} respectively, while these vertices represent six corresponding modules m_a-m_f of the placement illustrated in Figure 22. The module widths w_a, w_b, w_c, w_d, w_e and w_f are 3, 4, 3, 5, 2 and 3, respectively. Correspondingly, the module heights h_a, h_b, h_c, h_d, h_e and h_f are 5, 5 3, 2, 1 and 5 in turn. For instance, for m_a ($m_a \rightarrow m_b/m_c/m_e/m_f$), four directed weighted edges (v_a, v_b), (v_a, v_c), (v_a, v_e), (v_a, v_f) are added into H_{atcg}. The edge weights w_{ab}, w_{ac}, w_{ae} and w_{af} are 4, 3, 3 and 3, accordingly. In contrast, we connect v_a to v_d with a directed weighted edge (v_a, v_d), whose edge weight w_{ad} is 5, in V_{atcg} due to $m_a \uparrow m_d$. Moreover, it is worth mentioning that, due to $m_d \Uparrow m_e$, we need to set a directed weighted edge (v_d, v_e) not only in H_{atcg} but also in V_{atcg}. Because of the transitive closure feature of ATCG, a secondary edge (v_a, v_e) should be assigned in V_{atcg}. We repeat this process on the other modules until the relationships among all of six modules in Figure 22 are defined.

As depicted in Figure 21, from ATCG placement representation we can surely know that module m_b is on the right of m_a due to the existing directed edge (v_a, v_b) in H_{atcg}. Furthermore, since edge weight w_{ab} is greater than w_a, which denotes width of m_a, we ensure that modules m_a and m_b are separated from each other. Similarly, we know that m_e is set over m_b according to the weighted directed edge (v_b, v_e) in V_{atcg}. In contrast, the weighted directed edge (v_c, v_f) in V_{atcg} indicates that module m_c is possible to be partially merged with m_f vertically.

4.1.2. From an ATCG to the Corresponding Placement

We can derive the corresponding general placement from an ATCG in *O(nlgn)* time (*$O(n^2)$* time if any advanced constraints exist) by using a modified packing scheme based on the one initiated for TCG-S [46], where *n* is the number of modules. We use two red-black trees, which are denoted as RBT_h and RBT_v, to facilitate locating each module in the corresponding placement according to a packing sequence. During the packing process, the position of the latest packing module is primarily determined by the previously packed modules. Meanwhile, we need to consider any possible constraints and dual relations between the already packed modules and the newly being packed one.

Horizontal Packing

Figure 23 shows the packing flow with the aid of RBT_h. Each node of RBT_h represents a packing module. A newly being inserted node *n* denotes the current module *m* to be packed. The start node *s* in Line-1 represents the left boundary of a placement. In Line-3 every newly being packed module *m* is saved into an array for later comparison. Line-4 executes *InsertNode*RBT_h algorithm, whose basic principle is to insert the new node *n* to the right of a check node *p* in RBT_h under two situations. The first situation is when node *p* is the start node and its right node is NIL, while the other one is that nodes *p* and *n* have a horizontal relation and the right node of *p* is NIL. If there is a vertical relation between *p* and *n* and the left node of *p* is NIL, we set *n* to the left of *p*.

Algorithm: HorizontalPacking

Input: A feasible ATCG with packing-sequence

Output: X-coordinate of each module in the corresponding placement

Begin

1 insert the start node *s* into RBT_h

2 **for** (each packing module *m* from the packing-sequence)

3 save *m* into array *pakMod*

4 *InsertNodeRBTh*: checking RBT_h from the root node to find a proper position to insert the new node *n*, which is used to represent module *m* in RBT_h

5 **endfor**

End

Figure 23. Horizontal packing algorithm of ATCG.

The detailed pseudo-code of *InsertNodeRBT*$_h$ is illustrated in Figure 24. Lines 1-17 show the situation when the check node *p* is the start node in *RBT*$_h$. Line-7 indicates that the position of the newly being packed module *m* is firstly set by the module, which is represented as the *last* node before node *n* after an in-order traversal. Lines 8-9 direct that the bottom-left X-coordinate of *m* is updated by comparing it with the previously packed modules, which have horizontal constraints or relationship with the presently being processed module. Lines 13-16 imply that the top-right X-coordinate of the currently being packed module *m* is compared with the corresponding coordinates of all the modules, which are represented by all the nodes after node *n* in *RBT*$_h$. The one with smaller or the same value is to be deleted from *RBT*$_h$. When the check node *p* is not the start node, we need to check the relationship between *p* and the newly being inserted node *n* as pointed in Line-19. If there is a horizontal relation between *p* and *n*, it goes over the same process as before except that Line-22 notes the weight between *last* and *m* is added in order to properly position module *m* on the horizontal axis. If node *p* is vertical to the new node *n*, Lines 25-31 instruct a similar way to handle this situation, but the left node of *p* is employed instead of its right node.

Algorithm: InsertNodeRBT$_h$

Input: check node p in RBT_h and new node n

Output: X-coordinate of a newly inserted module m, which is represented by node n in RBT_h

Begin

1 **if** (p is the start node s)

2 **if** (right node of p is not a NIL node)

3 checking from the right node of p in RBT_h to find a proper position to insert n

4 **else**

5 set n as the right node of p

6 find the last previous node *last* before node n after an in-order traversal

7 set the top-right X-coordinate of *last* as the bottom-left X-coordinate of m

8 compare the bottom-left X-coordinate of m with the constraints between each previously packed module in *pakMod* and m, then the largest one is set as the bottom-left X-coordinate of m

9 compare the bottom-left X-coordinate of m with the bottom-left X-coordinate of the previously packed modules in *pakMod*, which has primary/secondary horizontal relation with m, plus the primary/secondary weight, then the largest one is set as the bottom-left X-coordinate of m

10 set the top-right X-coordinate of m as the sum of the bottom-left X-coordinate of m and width of m

11 balance RBT_h

12 do an in-order traversal of RBT_h and put all nodes after n into stack *nodes*

13 **while** (*nodes* is not empty)

14 compare the top-right X-coordinate of the top node saved in *nodes* with the top-right X-coordinate of m, then delete the node from RBT_h if with less or equal value

15 pop out the top node in *nodes*

16 **endwhile**

```
17   endif
18 else
19    check the relationship between nodes p and n
20    if (there is a primary/secondary horizontal relation between p and n)
21        do the same as Lines 2-6
22        use the top-right X-coordinate of last plus the primary/secondary weight
            of module m to update the bottom-left X-coordinate of module m
23        do the same as Lines 8-17
24    else
25        if (the left node of p is not a NIL node)
26            checking RBT_h from the left node of p to find a proper position to
              insert n
27        else
28            set n as the left node of p
29            find the last previous node last before n after an in-order traversal
30            do the same as Lines 22-23
31        endif
32    endif
33 endif
End
```

Figure 24. ATCG algorithm of node insertion to RBT_h.

Vertical Packing

Similarly, we can also determine the top-right Y-coordinate of all the packing modules by using RBT_v as shown in Figure 25. We use the start node *t* in Line-1 to represent the bottom boundary of a placement. Every currently being packed module *m* is saved into an array for later comparison as indicated in Line-3. Line-4 executes *InsertNodeRBTv* algorithm, which runs in a similar way to $InsertNodeRBT_h$ algorithm.

Algorithm: VerticalPacking

Input: A feasible ATCG with packing-sequence

Output: Y-coordinate of each module in the corresponding placement

Begin

1 insert the start node *t* into RBT_v

2 **for** (each packing module *m* from the packing-sequence)

3 save module *m* into array *pakMod*

4 *InsertNodeRBTv*: checking RBT_v from the root node to find a proper position to insert the new node *n*, which is used to represent module *m* in RBT_v

5 **endfor**

End

Figure 25. Vertical packing algorithm of ATCG.

Figure 26 illustrates the detailed pseudo-code of *InsertNodeRBT*$_v$. When a check node p in *RBT*$_v$ is a start node, the right node of p is checked. By checking if the right node of p is a NIL node, we will determine whether or not the new node n is inserted to the right of p as presented in Lines 1-17. The relationship of nodes p and n needs to be checked if p is not a start node. Lines 19-33 show how to handle two situations, i.e., a vertical or horizontal relation between p and n. After an in-order traversal, the *last* node before n is used to set the position of the newly being packed module m as indicated in Lines 7 and 22. The bottom-left Y-coordinate of m is updated by comparing it with the previously packed modules, which have horizontal constraints or relation with module m, as described in Lines 8-9. Lines 13-16 imply that the nodes representing the modules, which are covered by module m on the Y axis, are deleted from *RBT*$_v$. When p is horizontal to the new node n, the left node of p is utilized to deal with this condition in a similar way as shown in Lines 25-31.

Algorithm: InsertNodeRBT$_v$

Input: check node p in RBT_v and new node n

Output: Y-coordinate of a newly inserted module m, which is represented by node n in RBT_v

Begin

1 **if** (p is the start node t)

2 **if** (the right node of p is not a NIL node)

3 checking RBT_v from the right node of p to find a proper position to insert node n

4 **else**

5 set n as the right node of p

6 find the last previous node *last* before n after an in-order traversal

7 set the top-right Y-coordinate of *last* as the bottom-left Y-coordinate of module m

8 compare the bottom-left Y-coordinate of m with the constraints between each previously packed module in *pakMod* and m, then the largest one is set as the bottom-left Y-coordinate of m

9 compare the bottom-left Y-coordinate of m with the bottom-left Y-coordinate of the previously packed modules in *pakMod*, which has primary/secondary vertical relation with m, plus the primary/secondary weight, then the largest one is set as the bottom-left Y-coordinate of m

10 set the top-right Y-coordinate of m as the sum of the bottom-left Y-coordinate of m and width of m

11 balance RBT_v

12 do an in-order traversal RBT_v and put all nodes after n into stack *nodes*

13 **while**(*nodes* is not empty)

14 compare the top-right Y-coordinate of the top node saved in *nodes* with the top-right Y-coordinate of m, then delete the node from RBT_v if with less or equal value

15 pop out the top node in *nodes*

16 **endwhile**

```
17   endif
18 else
19    check the relationship between nodes p and n
20    if (there is a primary/secondary vertical relation between p and n)
21          do the same as Lines 2-6
22          use the top-right Y-coordinate of last plus the primary/secondary
             weight of module m to update the bottom-left Y-coordinate of m
23          do the same as Lines 8-17
24    else
25         if (the left node of p is not a NIL node)
26            checking RBT_v from the left node of p to find a proper position to
               insert n
27         else
28            set n as the left node of p
29            find the last previous node last before n after an in-order traversal
30            do the same as Lines 22-23
31         endif
32    endif
33 endif
End
```

Figure 26. ATCG algorithm of node insertion to RBT_v .

4.1.3. Properties of ATCG

There are three properties for a feasible ATCG:

Property-1: H_{atcg} and V_{atcg} are acyclic.

Property-2: Each pair of vertices must be connected by one or two edges. If both horizontal and vertical edges connect a pair of vertices, there should be a dual relation between the two corresponding modules.

Property-3: The transitive closure of H_{atcg}/V_{atcg} is equal to itself.

Proof: ATCG is based on the original TCG, which must be acyclic. The secondary edges are added only if they do not form a cycle in the constraint graphs. Thus, Property-1 holds.

We construct edges to reflect all geometric relations among modules in a placement. Following the property of the original TCG, there must be exactly one horizontal or vertical edge, i.e., the primary edge in ATCG, between each two vertices. Based on this, the secondary edge in ATCG cannot be the same as the primary edge. Two different edges together can represent two-directional relationship, i.e., dual relation. Thus, Property-2 is proved.

ATCG follows the original TCG, which has the same property-3 as well. The added secondary edges must not form a cycle in the constraint graphs but preserve the transitive closure feature. Therefore Property-3 still holds. Q.E.D.

Based on the properties above, we have the following theorem:

Theorem. There exists a unique placement corresponding to an ATCG.

Proof: Below we first show that each ATCG is feasible (i.e., there must exist a placement for each ATCG), and then show the uniqueness of the placement. Property-1 ensures that no module is both on the left and on the right to (or below and above) another module in the packing. Property-2 guarantees the accuracy of the resultant packing. Thus, Property-1 and Property-2 ensure that there exists a placement for each ATCG. Given an ATCG, the X and Y coordinates of each module are determined by the respective longest paths in the horizontal and vertical constraints graphs, where edge weights are well defined. Therefore, the placement is unique. Q.E.D.

4.2. Placement Algorithm

Since the proposed ATCG representation is based on the plain TCG, the developed symmetry-aware theory and practice for SP-TCG in Chapter 3 can be directly used for ATCG-based placement algorithm construction. In this regard, we still use the conventional simulated annealing (SA) algorithm to perturb placement configurations, although genetic algorithm (GA) may offer better performance than SA, so as to compare with previous methods in the literature most of which use SA as the search engine. Before building a valid ATCG, we need to construct a feasible TCG. Instead of the conventional reduction-edge-based methods [44], in this work we have leveraged SP to facilitate the internal TCG operations as described in Chapter 3. In this way, the time complexity can be reduced from

$O(n^2)$ to $O(n)$, where n is the number of modules. During the random TCG perturbation, we employ the perturbation redundancy control as introduced in Section 3.4, to prevent unnecessary repetition of the same states.

After any operation of the plain TCG as originally defined in [44], any secondary edges (i.e., for dual relations) would be added to form a feasible ATCG. Then the packing operation as described in Section 4.1.2 is conducted, and the resultant cost is compared with that of the previous state to decide how to continue the search in the SA optimization process. The following lemma can ensure how to form a valid ATCG.

Lemma. Any secondary edge can be constructively added to transform a TCG to a valid ATCG.

Proof: Without loss of generality, we consider there is an edge from v_i to v_k in the horizontal TCG. Below with proof by contradiction, we will first prove adding an edge from v_i to v_k in the vertical TCG would not form a cyclic graph. Assume this proposition is false, that is equivalent to say, there exists a vertex v_j in the vertical TCG that include one edge from v_k to v_j and another edge from v_j to v_i. Based on the TCG transitive property, there should already have existed an edge from v_k to v_i in the vertical TCG. But this is a contradiction to the given context that an edge from v_i to v_k exists in the horizontal TCG while there is one and only one edge between two vertices in TCG. Therefore, we can ensure that adding an edge from v_i to v_k in the vertical TCG keeps the graph acyclic. By adding edges from the fan-in vertices of v_i to v_k and from v_i to the fan-out vertices of v_k in the vertical TCG, we can construct valid ATCG that meets the three properties described in Section 4.1.3. Q.E.D.

4.3. Experimental Results

We have developed an SA-based placement algorithm for analog layout designs by using the proposed ATCG method. Different from TCG-S [46], we maintain SP (i.e., both α- and β-sequences) in amortized constant time to aid TCG operations. According to the lemma presented in Section 4.2, we can randomly derive a valid ATCG based on the updated TCG during the SA iteration. With the packing-sequence (i.e., the β-sequence of the corresponding SP), we can obtain the actual placement of the ATCG by following the flow of Figures 23 and 25.

To test the performance of our proposed algorithm, it has been coded in C++ and applied to several test circuits on an Intel X86 1.2GHz Linux workstation that has 64GB of memory. To demonstrate the effectiveness of our proposed ATCG method, two other approaches coded in C++ on the same platform have been included for comparison: (i)AbsPlace, one absolute placement scheme using absolute coordinates [54]; and (ii) HB*-trees, an implementation that imitates [20][21]. The comparison results are given in Table V, where less packing cost (denoted "*PackCost*") and less execution time (denoted "*T (sec.)*") are preferable. The data shows that our proposed ATCG method outperforms AbsPlace in terms of search quality (i.e., PackCost), and achieves similar performance compared to HB*-trees. However, it should be noted that the advanced spacing/merging constraints cannot be considered in HB*-trees, different from ATCG and AbsPlace. On

average, compared with HB*-trees, ATCG spent slightly more CPU time by 4.68%. But in comparison with AbsPlace, ATCG reduces cost by 23.17% and execution time by 36.66% on average.

Table V. Comparison of alternative methods on three test circuits.

Test Circuits		*AbsPlace*	*HB*-Trees*	*ATCG*
Circuit-1	PackCost	6.96e+07	5.46e+07	5.74e+07
	T (sec.)	1.32	0.95	0.87
Circuit-2	PackCost	9.09e+05	6.93e+05	7.03e+05
	T (sec.)	9.06	4.99	5.76
Circuit-3	PackCost	1.50e+06	1.02e+06	1.06e+06
	T (sec.)	43.61	24.00	26.40

4.4. Summary

In this chapter, we proposed ATCG placement representation to facilitate the handling of analog layout design. After reviewing different works from the literature, we presented how to construct an ATCG and derive its corresponding placement. Then we

presented the details of our proposed packing method, which is based on red-black tree data structure. We also proved the properties and validity of ATCG. Our experimental results showed that this proposed representation can help efficiently generate high-performance placement in compliance with complex analog constraints.

The next chapter will provide the conclusion and future scope of this work.

5 Conclusions and Future Work

In this chapter we will review the entire research work and then the principal contributions of this thesis will be summarized. Moreover, we recommend several interesting topics for the future research work. Finally, a list of our published papers will be provided along with their links to the specific chapter within this thesis.

In this thesis, we have reviewed different general placement methods and representations. After introducing the analog placement problem, we have surveyed various techniques to glance over how they have tackled analog placement over the time from old VLSI technologies to the advanced nanometer technologies. Then we have discussed the advantages and limitations of these techniques as well. Base on those previous works, we have presented our proposed SP-driven TCG approach and shown that it can effectively and efficiently deal with analog placement in the context of symmetry constraints. Furthermore, we have suggested ATCG topological representation to facilitate the handling of analog layout design. In the meantime, the SP-driven TCG technique is embraced in the ATCG placement method. Our obtained experimental results have also demonstrated the superiority of the proposed ATCG placement representation.

5.1 Contributions of the Thesis

The major contributions of the thesis are listed below:

- The thesis presents a sequence-pair (SP) driven transitive-closure-graph (TCG) method to deal with analog placement, which is an indispensable stage in the analog IC layout design. The SP-driven TCG has promising benefit in handling advanced analog constraints.
- Focusing on the symmetry-aware placement, the thesis defines three special symmetric-feasible conditions and describes the construction of symmetric placement with our proposed SP-driven TCG method.
- A series of SP-driven perturbation operations are proposed in the thesis. SP are utilized to guide the perturbation operations of a symmetric-feasible TCG. Moreover, we prove that the symmetric-feasible conditions and inherent properties of SP can guarantee symmetric feasibility and validity of TCG after the random perturbation process.
- A redundancy control scheme is suggested when forming the representation states to generate high-performance analog placement with satisfactory computation efficiency. The experimental results demonstrate the efficacy of the proposed method.
- The thesis proposes an advanced transitive-closure-graph-based placement representation (ATCG) to facilitate analog layout design. The versatility and flexibility of ATCG can ensure it to accurately control spacing and merging constraints uniquely required by analog layout design.

- The construction of a valid ATCG is detailed in the thesis, while the corresponding placement can be derived by our modified packing scheme based on the red-black tree data structure. Furthermore, the thesis proves the properties and validity of a feasible ATCG. The experimental results showed that this proposed representation can efficiently assist in generating high-performance analog placement with high computation efficiency.

5.2 Scope of the Future Work

This research work provides a solid base for the future work in order to deal with even more complicated constraints emerging from the analog layout design. The ATCG representation developed in this thesis can be extended to handle other various geometric constraints and advanced design requirements, which are necessary in the modern IC designs to satisfy stringent performance specifications. While this thesis itself is focused on the feasibility study of innovative topological representations and the theoretical correctness of placement operations, their broad applications in the analog layouts still need more thorough performance verification through the analog module generation, interconnect routing, and post-layout simulation from the completed layouts.

More recently, some preliminary research has been conducted on the applications of machine learning techniques to the analog physical design automation, including smart well generation [55], quality prediction [56], symmetry detection [57], placement [58][59], and routing [60]. For the placement study using machine learning techniques, the modern

research is still limited to the absolute coordinates based representation or indirect mediums. We expect the topological representations should do a good job in this regard. So it is worthwhile for us to continue this study to explore how the proposed ATCG representation and the associated operation can be involved in the machine-learning-based placement algorithms.

Moreover, analog placement design will face more challenges in the near future as the emergence of the state-of-the-art FinFET technology and the next-generation carbon nanotube field-effect transistor (CNTFET) technology [61]. The possibility and potentiality of our current research work can be further exploited to be compatible with these new technologies. Eventually, it is expected that the outcome of the advanced analog placement research work would be widely and deeply applied in the commercial products for industry-level analog circuit layout design.

5.3 The Candidate's Published Papers

[1] L. He, Z. Zhao, Y. Chen, and L. Zhang, "Advanced Transitive-Closure-Graph-Based Placement Representation for Analog Layout Design," in *Proc. 27rd IEEE International Conference on Electronics Circuits and Systems (ICECS)*, virtually (originally planned to take place in Glasgow, Scotland), Nov. 23-25, 2020.

This paper is out of the detailed research work documented in Chapter 4 of this thesis.

[2] L. He, Z. Zhao, Y. Chen, and L. Zhang, "Placement with Sequence-Pair-Driven TCG for Advanced Analog Constraints," in *Proc. 33rd IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, virtually (originally planned to take place in London, ON, Canada), Aug. 30-Sept. 02, 2020.

This paper is out of the detailed research work documented in Chapter 3 of this thesis.

[3] L. He and Y. Chen, "A Proximity-Aware Representation for Placement Problems in IC Design," in *Proc. 23rd Annual Newfoundland Electrical and Computer Engineering Conference (NECEC)*, Nov. 2014.

This paper is out of the detailed research work documented in Chapters 1-2 of this thesis.

Bibliography

[1] X. Dong and L. Zhang, "EA-Based LDE-Aware Fast Analog Layout Retargeting with Device Abstraction," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 27, no. 4, pp. 854-863, 2019.

[2] T. Liao and L. Zhang, "Analog Integrated Circuit Sizing and Layout Dependent Effects: A Review," *Journal of Microelectronics and Solid State Electronics*, vol. 3, no. 1A, pp. 17-29, 2014.

[3] D. Jepsen and C. Gelatt, "Macro Placement by Monte Carlo Annealing," in *Proc. IEEE International Conference on Computer Design*, pp. 495-498, Nov. 1983.

[4] W. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 349-359, Mar. 1995.

[5] J. Cohn, D. Garrod, R. Rutenbar, and L. Carley, "KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing," *IEEE Journal of Solid-State Circuits*, vol. 26, no.3, pp. 330-342, Mar. 1991.

[6] K. Lampaert, G. Gielen, and W. Sansen, Analog Layout Generation for Performance and Manufacturability. Boston, MA: Kluwer, 1999.

[7] L. Zhang, R. Raut, Y. Jiang, and U. Kleine, "Placement Algorithm in Analog Layout Designs," IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, vol. 25, no. 10, pp. 1889-1903, 2006.

[8] L. Zhang, U. Kleine, R. Raut, and Y. Jiang, "An Automated Design Tool for Analog Layouts," IEEE Transactions on Very Large Scale Integration Systems, vol. 14, no. 8, pp. 881-894, 2006.

[9] H. Ou, K. Tseng, J. Liu, I. Wu and Y. Chang, "Layout-Dependent Effects-Aware Analytical Analog Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, pp. 1243-1254, Aug. 2016.

[10] R. H. J. M. Otten, "Automatic Floorplan Design," in *Proc. Design Automation Conference*, pp. 261-267, 1982.

[11] D. F. Wong and C. L. Liu, "A New Algorithm for Floorplan Design," in *Proc. Design Automation Conference*, pp. 101-107, 1986.

[12] M. Lin, Y. Chang, and C. Hung, "Recent Research Development and New Challenges in Analog Layout Synthesis," in *Proc. IEEE Asia and South-Pacific Design Automation Conference*, pp. 617-622, 2016.

[13] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no.12, pp. 1518-1524, Dec. 1996.

[14] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module Packing Based on the BSG-Structure and IC Layout Applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 519-530, Jun. 1998.

[15] P. Guo, C. Cheng, and T. Yoshimura, "An O-tree Representation of Nonslicing Floorplan and Its Applications," in *Proc. Design Automation Conference*, pp. 268-273, June 1999.

[16] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu, "B*-Trees: A New Representation for Nonslicing Floorplans," in *Proc. Design Automation Conference*, pp. 458-463, June 2000.

[17] X. Hong, G. Huang, T. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu, "Corner Block List: An Effective and Efficient Topological Representation of Nonslicing Floorplan," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp. 8-12, Nov. 2000.

[18] J. Lin and Y. Chang, "TCG: A Transitive Closure Graph-Based Representation for General Floorplans," *IEEE Transactions on VLSI Systems*, vol. 13, no.2, pp. 288-292, Feb. 2005.

[19] J. Lin and Y. Chang, "TCG-S: Orthogonal Coupling of P*-Admissible Representations for General Floorplans," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no.6, pp. 968-980, June 2004.

[20] P. Lin, Y. Chang, and S. Lin, "Analog Placement Based on Symmetry-Island Formulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no.6, pp.791-804, June 2009.

[21] I. Wu, H. Ou, and Y. Chang, "QB-Trees: Towards An Optimal Topological Representation and Its Applications to Analog Layout Designs," in *Proc. ACM/IEEE Design Automation Conference*, no. 80, pp 1-6, June 2016.

[22] Y. Lu, Y. Chang, and Y. Chang, "WB-Trees: A Meshed Tree Representation for FinFET Analog Layout Designs," in *Proc. ACM/IEEE Design Automation Conference*, no. 9, pp 1-6, June 2018.

[23] [Online]. Available: https://slidesplayer.org/slide/15483067/

[24] J. Cohn, D. Garrod, R. Rutenbar, and L. Carley, *Analog Device-Level Layout Automation*. Boston: Kluwer Academic Publishers, 1994.

[25] ICsense-The IC Design Company. [Online]. Available: https://www.icsense.com/

[26] [Online]. Available: https://www.electronicsweekly.com/news/design/eda-and-ip/mixed-signal-changing-chip-design-in-europe-says-cadence-2013-12/

[27] N. Sherwani, *Algorithms for VLSI Physical Design Automation*. Norwell: Kluwer Academic Publisher, 1999.

[28] A. Kahng and Q. Wang, "Implementation and Extensibility of An Analytic Placer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no.5, pp. 734-747, May 2005.

[29] A. Caldwell, A. Kahng, and I. Markov, "Can Recursive Bisection Alone Produce Routable Placements?" in *Proc. 37th Design Automation Conference*, pp. 477-482, June 2000.

[30] L. Zhang and U. Kleine, "A Genetic Approach to Analog Module Placement with Simulated Annealing," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 345-348, May 2002.

[31] M. Strasser, M. Eick, H. Graeb, U. Schlichtmann, and F. Johannes, "Deterministic Analog Circuit Placement Using Hierarchically Bounded Enumeration and Enhanced Shape Functions," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp. 306-313, 2008.

[32] E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, "Automation of IC Layout with Analog Constraints," *IEEE Transactions on Computer-Aided Design Integrated Circuits System*, vol. 15, no.8, pp. 923-942, Aug. 1996.

[33] K. Lampaert, G. Gielen, and W. Sansen, "A Performance-Driven Placement Tool for Analog Integrated Circuits," *IEEE Journal of Solid-State Circuits*, vol. 30, no.7, pp. 773-780, Jul. 1995.

[34] H. Lee, Y. Chang, and H. Yang, "MB*-Tree: A Multilevel Floorplanner for Large-Scale Building-Module Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no.8, pp. 1430-1444, Aug. 2007.

[35] Rob A Rutenbar, "Design Automation for Analog: the Next Generation of Tool Challenges," in *Proc. IEEE/ACM International Conference on Computer-aided Design*, pp. 458–460, 2006.

[36] R. Rutenbar, J. Cohn, and M. Lin, *Layout Tools for Analog Integrated Circuits and Mixed-Signal Systems on Chip: A Survey*. Chapter 16 in EDA for IC Implementation, Circuit Design and Process Technology, 2nd edition, Luciano Lavagno, Grant Martin and Louis Scheffer, eds., CRC Press, 2016.

[37] H. Graeb, *Analog Layout Synthesis: A Survey of Topological Approaches*. Berlin: Springer, 2012.

[38] M. Lin, P. Chang, S. Lee, and H. Graeb, "DeMixGen: Deterministic Mixed-Signal Layout Generation with Separated Analog and Digital Signal Paths," *IEEE Transactions on Computer-Aided Design Integrated Circuits System*, vol. 35, no.8, pp. 1229-1242, 2016.

[39] L. Zhang, R. Raut, Y. Jiang, U. Kleine, and Y. Kim, "Macro-Cell Placement for Analog Physical Designs using a Hybrid Genetic Algorithm with Simulated Annealing," *International Journal of Integrated Computer-Aided Engineering*, vol. 12, no. 4, pp. 379-396, 2005.

[40] F. Balasa and K. Lampaert, "Symmetry within the Sequence-Pair Representation in the Context of Placement for Analog Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no.7, pp. 721-731, July 2000.

[41] S. Koda, C. Kodama, and K. Fujiyoshi, "Linear Programming-Based Cell Placement with Symmetry Constraints for Analog IC Layout," *IEEE Transactions on Computer-Aided Design Integrated Circuits System*, vol. 26, no. 4, pp. 659-668, April 2007.

[42] Y. Pang, F. Balasa, K. Lampaert, and C. Cheng, "Block Placement with Symmetry Constraints based on the O-tree Non-Slicing Representation," in *Proc. 37th Design Automation Conference*, pp. 464-467, 2000.

[43] F. Balasa, S. Maruvada, and K. Krishnamoorthy, "On the Exploration of the Solution Space in Analog Placement with Symmetry Constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 2, pp.177-191, Feb. 2004.

[44] L. Zhang, C.-J. R. Shi, and Y. Jiang, "Symmetry-Aware Placement with Transitive Closure Graphs for Analog Layout Design," in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conference*, pp. 180-185, Jan. 2008.

[45] R. He and L. Zhang, "Symmetry-Aware TCG-Based Placement Design under Complex Multi-Group Constraints for Analog Circuit Layouts," in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conference*, pp. 299-304, 2010.

[46] J. Lin, G. Wu, Y. Chang, and J. Chuang, "Placement with Symmetry Constraints for Analog Layout Design Using TCG-S," in *Proc. IEEE/ACM Asia and South-Pacific Design Automation Conference*, pp. 1135-1138, 2005.

[47] Y. Tam, E. Young, and C. Chu, "Analog Placement with Symmetry and Other Placement Constraints, Computer-Aided Design," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp. 349-354, 2006.

[48] L. Xiao and E. Young, "Analog Placement with Common Centroid and 1-D Symmetry Constraints," in *Proc. Asia and South-Pacific Design Automation Conference*, pp. 353-360, 2009.

[49] Q. Ma, L. Xiao, Y. Tam, and E. Young, "Simultaneous Handling of Symmetry, Common Centroid, and General Placement Constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp.85-95, Jan. 2011.

[50] H. Tsao, P. Chou, S. Huang, Y. Chang, M. Lin, D. Chen, and D. Liu, "A Corner Stitching Compliant B∗-Tree Representation and Its Applications to Analog Placement," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp. 507-511, 2011.

[51] P. Wu, M. Lin, X. Li, T. Ho, “Parasitic-Aware Common-Centroid FinFET Placement and Routing for Current-Ratio Matching,” *ACM Trans. Design Automation Electronic Systems*, vol. 21, no. 3, pp. 39:1-39:22, 2016.

[52] M. Lin, V. Hsiao, C. Lin, and N. Chen, “Parasitic-Aware Common-Centroid Binary-Weighted Capacitor Layout Generation Integrating Placement, Routing, and Unit Capacitor Sizing,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 8, pp. 1274-1286, 2017.

[53] T. Wang, P. Wu, M. Lin, “Automatic Adaptive MOM Capacitor Cell Generation for Analog and Mixed-Signal Layout Design,” in *Proc. IEEE/ACM Design Automation Conference*, pp. 1-2 (late breaking results), 2020.

[54] L. Zhang, “VLSI Circuit Layout,” in *Wiley Encyclopedia of Computer Science and Engineering*, vol. 5, pp. 3034-3044, 2009.

[55] B. Xu, Y. Lin, X. Tang, S. Li, L. Shen, N. Sun, D. Pan, “WellGAN: Generative-Adversarial-Network-Guided Well Generation for Analog/Mixed-Signal Circuit Layout,” in *Proc. 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1-6, Las Vegas, NV, USA, 2019.

[56] M. Liu, K. Zhu, J. Gu, L. Shen, X. Tang, N. Sun, and D. Pan, “Towards Decrypting the Art of Analog Layout: Placement Quality Prediction via Transfer Learning,” in

Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 496-501, Grenoble, France, 2020.

[57] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N, Sun, D. Pan, "S^3DET: Detecting System Symmetry Constraints for Analog Circuits with Graph Similarity," in *Proc. 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 193-198, Beijing, China, 2020.

[58] J. Rosa, D. Guerra, N. Horta, R. Martins, N. Lourenço, *Using Artificial Neural Networks for Analog Integrated Circuit Design Automation*. Springer International Publishing, 2020.

[59] Y. Li, Y. Lin, M. Madhusudan, A. Sharma, W. Xu, S. Sapatnekar. R. Harjani, and J. Hu, "Exploring a Machine Learning Approach to Performance Driven Analog IC Placement," in *Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* pp. 24-29, Limassol, Cyprus, 2020.

[60] K. Zhu, M. Liu, Y. Lin, B. Xu, S. Li, X. Tang, N. Sun, D. Pan., "GeniusRoute: A New Analog Routing Paradigm Using Generative Neural Network Guidance," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1-8, Westminster, CO, USA, 2019.

[61] G. Hills, C. Lau, A. Wright, et al., “Modern Microprocessor Built from Complementary Carbon Nanotube Transistors,” *Nature*, 572, pp. 595–602, 2019.