# A MODIFIED RELAY-RACE APPROACH TO AUTOMATED FLOORPLANNING IN PCB AND IC DESIGN

by

Mert Vatansever

B.S., Electronics and Communication Engineering, Istanbul Technical University, 2016

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2018

A MODIFIED RELAY-RACE APPROACH TO AUTOMATED FLOORPLANNING IN PCB AND IC DESIGN

APPROVED BY:

Assist. Prof. Faik Başkaya .....
(Thesis Supervisor)

Assist. Prof. Engin Afacan .....

Prof. Günhan Dündar .....

DATE OF APPROVAL: 02.08.2018

# ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor, Assist. Prof. Faik Başkaya, for his continuous support, suggestions and patience. His guidance and knowledge of physical design automation helped me in all time of research and writing of this thesis.

In addition, I would also like to thank my thesis committee, Prof. Günhan Dündar and Assist. Prof. Engin Afacan for their insightful suggestions and comments.

Lastly, I would like to thank my father, Mümün Vatansever, my mother, Neşe Vatansever, and my brother, Furkan Hakan Vatansever, for their constant support and encouragement.

# ABSTRACT

## A MODIFIED RELAY-RACE APPROACH TO AUTOMATED FLOORPLANNING IN PCB AND IC DESIGN

Floorplanning is a fundamental design step in the physical design of IC and PCB, as it manages the complexity of circuit design, which is increasing with the progression in technology. Floorplanning optimizes the locations of the modules to reduce the circuit area and wirelength of interconnections. Therefore, floorplanning provides a base assignment for circuit layout. From the computational point of view, floorplanning problem is an NP hard problem. The size of the search space grows exponentially with increasing number of modules. Thus, it is a very challenging task to find an optimum solution. The scope of the search space and the complexity of transformation between floorplanning representation and its corresponding floorplan are determined by the representation method. Researchers have proposed many representation methods such as Polish Notation, Bounded Sliced Grid (BSG), Transitive Closure Graph (TCG), B*Tree, and Sequence Pair (SP). In addition to the representation method, floorplanning algorithm is another essential factor for speed and quality of the floorplanning process. There are various successful stochastic algorithms for floorplanning including Simulated Annealing (SA), Genetic Algorithm (GA), and Relay Race Algorithm (RRA). However, there are also shortcomings of these traditional algorithms. In this thesis, a Modified Relay Race Approach (MRRA) is proposed to overcome these shortcomings. It utilizes basic methods of RRA such as rough search, focusing search, and relay. On the other hand, it uses dual path search method and better optimized parameters to improve the search ability and run time. Based on the experimental results utilizing MCNC benchmarks, MRRA increased the solution quality and decreased the run time of area optimization, comparing with SA, GA and RRA.

# ÖZET

## PCB VE IC DEVRELERİN OTOMATİK YERLEŞİM PLANLAMASI İÇİN MODİFİYE EDİLMİŞ BAYRAK YARIŞI YÖNTEMİ

Yerleşim planlaması, teknolojideki ilerlemelerle birlikte artan devre tasarımı zorluğunu yöneterek, IC ve PCB fiziksel tasarımlarında temel bir tasarım adımını temsil etmektedir. Yerleşim planlaması, devrenin alanını ve modüller arasındaki bağlantıların toplam uzunluğunu azaltmak için modüllerin yerlerini optimize etmektedir. Bu nedenle yerleşim planlaması, devre yerleşimi için bir zemin hazırlamaktadır. Arama uzayının boyutu, modüllerin sayısındaki artış ile birlikte katlanarak büyümektedir. Yerleşim planlaması gösterimi ile bu gösterime karşılık gelen yerleşim planı arasındaki dönüşümün karmaşıklığı gösterim yöntemi ile belirlenmektedir. Araştırmacılar, Polish Notation, Bounded Slicing Grid (BSG), Transitive Closure Graph (TCG), B*Tree ve Sequence Pair (SP) gibi birçok gösterim yöntemini önermişlerdir. Yerleşim planlaması algoritması, yerleşim planı sürecinin hızı ve kalitesi için bir başka önemli faktördür. Simulated Annealing (SA), Genetic Algorithm (GA) ve Relay Race Algorithm (RRA), yerleşim planı problemi için araştırmacılar tarafından kullanılan en popüler stokastik algoritmalar arasında bulunmaktadır. Fakat bu geleneksel algoritmaların da eksiklikleri bulunmaktadır. Bu tezde, bu algoritmaların eksikliklerinin üstesinden gelmek için Modified Relay Race Approach (MRRA) önerilmektedir. RRA tarafından kullanılan kaba arama, detaylı arama ve mutasyon gibi yöntemler, MRRA tarafından da kullanılmaktadır. Öte yandan MRRA, arama yeteneğini ve hesaplama süresini geliştirmek için çift yönlü arama yöntemini ve daha iyi optimize edilmiş parametre değerlerini kullanmaktadır. MRRA, MCNC kriterleri kullanılarak yapılan deneylerin sonuçları göz önünde bulundurulduğunda SA, GA ve RRA yaklaşımlarına göre alan optimizasyonuna ait çözüm kalitesini arttırırken hesaplama süresini de azaltmaktadır.

# TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . . iii
ABSTRACT . . . . . iv
ÖZET . . . . . v
LIST OF FIGURES . . . . . vii
LIST OF TABLES . . . . . ix
LIST OF SYMBOLS . . . . . xi
LIST OF ACRONYMS/ABBREVIATIONS . . . . . xii
1. INTRODUCTION . . . . . 1
1.1. Physical Design Process . . . . . 2
1.2. Floorplanning Problem . . . . . 3
1.3. Representation of Floorplan . . . . . 5
1.3.1. Polish Notation . . . . . 6
1.3.2. Bounded Slicing Grid . . . . . 7
1.3.3. Transitive Closure Graph . . . . . 8
1.3.4. B*Tree . . . . . 9
1.3.5. Sequence Pair . . . . . 10
2. EXISTING FLOORPLANNING APPROACHES . . . . . 11
2.1. Simulated Annealing . . . . . 12
2.2. Genetic Algorithm . . . . . 16
2.3. Relay Race Algorithm . . . . . 20
3. MODIFIED RELAY RACE APPROACH . . . . . 25
3.1. Overview . . . . . 25
3.2. Moving Methods . . . . . 28
3.3. Cost Function . . . . . 32
4. EXPERIMENTS AND RESULTS . . . . . 33
4.1. Parameter Setting . . . . . 33
4.2. Comparison . . . . . 42
5. CONCLUSION AND FUTURE WORK . . . . . 46
REFERENCES . . . . . 47

# LIST OF FIGURES

Figure 1.1. Physical Design Process. . . . . 2

Figure 1.2. Types of Floorplan . . . . . 4

Figure 1.3. Polish Notation and Its Corresponding Floorplan . . . . . 6

Figure 1.4. BSG and Its Corresponding Floorplan . . . . . 7

Figure 1.5. TCG and Its Corresponding Floorplan . . . . . 8

Figure 1.6. B*Tree and Its Corresponding Floorplan . . . . . 9

Figure 1.7. Modules and floorplan of the SP $(\Gamma+, \Gamma-) = (abdecf, cbfade)$ . . . 10

Figure 2.1. Flowchart of Simulated Annealing . . . . . 14

Figure 2.2. Flowchart of Genetic Algorithm . . . . . 18

Figure 2.3. Basic Concept of Relay Race Algorithm . . . . . 21

Figure 2.4. Rough search and focusing search . . . . . 22

Figure 2.5. Flowchart of Relay Race Algorithm . . . . . 22

Figure 2.6. Behavior of Relay Race Algorithm in the Solution Space . . . . . 24

Figure 3.1. Dual path search . . . . . 26

Figure 3.2. Flowchart of Modified Relay Race Algorithm . . . . . 26

Figure 3.3. Rotation Move . . . . . 28

Figure 3.4. Exchange Move . . . . . 29

Figure 3.5. Insertion Move . . . . . 30

Figure 4.1. Average cost investigation of combinations $N_f$ & $N_t$ . . . . . 37

Figure 4.2. Average run time investigation of combinations $N_f$ & $N_t$ . . . . . 37

Figure 4.3. Average cost investigation of combinations $N_r$ & $N_f$ & $N_t$ . . . . . 41

Figure 4.4. Average run time investigation of combinations $N_r$ & $N_f$ & $N_t$ . . 41

Figure 4.5. Comparison in the average cost . . . . . 44

Figure 4.6. Comparison in the average run time . . . . . 45

# LIST OF TABLES

Table 1.1. Comparison of different floorplan representations . . . . . 5

Table 4.1. The details of MCNC benchmark circuits . . . . . 33

Table 4.2. Results of RRA $N_r = 100$ & $N_f = 1000$ & $N_t = 10$ . . . . . 34

Table 4.3. Trial experiments for $N_f = 10.N_r$ & $N_t = 10$ $(N_r = 3.N_m)$ . . . . . 35

Table 4.4. Trial experiments for $N_f = 5.N_r$ & $N_t = 15$ $(N_r = 3.N_m)$ . . . . . 35

Table 4.5. Trial experiments for $N_f = 5.N_r$ & $N_t = 20$ $(N_r = 3.N_m)$ . . . . . 35

Table 4.6. Trial experiments for $N_f = 3.N_r$ & $N_t = 15$ $(N_r = 3.N_m)$ . . . . . 36

Table 4.7. Trial experiments for $N_f = 3.N_r$ & $N_t = 20$ $(N_r = 3.N_m)$ . . . . . 36

Table 4.8. Trial experiments for $N_f = 3.N_r$ & $N_t = 25$ $(N_r = 3.N_m)$ . . . . . 36

Table 4.9. Trial experiments for $N_r = 3.N_m$ & $N_f = 4.N_r$ & $N_t = 15$ . . . . . 39

Table 4.10. Trial experiments for $N_r = 2.N_m$ & $N_f = 4.N_r$ & $N_t = 20$ . . . . . 39

Table 4.11. Trial experiments for $N_r = 2.N_m$ & $N_f = 3.N_r$ & $N_t = 25$ . . . . . 39

Table 4.12. Trial experiments for $N_r = 1.N_m$ & $N_f = 5.N_r$ & $N_t = 40$ . . . . . 40

Table 4.13. Trial experiments for $N_r = 1.N_m$ & $N_f = 4.N_r$ & $N_t = 45$ . . . . . 40

Table 4.14. Trial experiments for $N_r = 1.N_m$ & $N_f = 3.N_r$ & $N_t = 50$ . . . . . 40

Table 4.15. Comparison in the average cost and average run time . . . . . 43

Table 4.16. Comparison in the best cost and worst cost . . . . . 43

Table 4.17. Improvement in the average cost and run time . . . . . 43

# LIST OF SYMBOLS

| | |
|---|---|
| $C_t$ | Total cost function |
| $C_a$ | Cost function of area |
| $C_w$ | Cost function of wirelength |
| $C_o$ | Cost function of overlap |
| $N_f$ | Maximum number of continuous trials without improvement during focusing run |
| $N_m$ | Number of modules in the circuit |
| $N_r$ | Maximum number of trials without improvement during rough run |
| $N_t$ | Total number of runners in the team for the relay |
| $R_e$ | Percentage of randomly generated part of the solution during relay operation |
| $\Gamma+$ | Positive Sequence |
| $\Gamma-$ | Negative Sequence |

# LIST OF ACRONYMS/ABBREVIATIONS

| | |
|---|---|
| BSG | Bounded Slicing Grid |
| CGP | Constraint Graph Pair |
| IC | Integrated Circuit |
| GA | Genetic Algorithm |
| HCG | Horizontal Constraint Graph |
| HPWL | Half Perimeter Wirelength |
| HTCG | Horizontal Transitive Closure Graph |
| MCNC | Microelectronics Center of North Carolina |
| MRRA | Modified Relay Race Approach |
| PCB | Printed Circuit Board |
| RRA | Relay Race Algorithm |
| SA | Simulated Annealing |
| SP | Sequence Pair |
| TCG | Transitive Closure Graph |
| VCG | Vertical Constraint Graph |
| VTCG | Vertical Transitive Closure Graph |

# 1. INTRODUCTION

The number of components in a circuit and the interconnections between these components increase rapidly as the technology improves over time [1]. Floorplanning optimizes the relative layout locations of the components to reduce the area of circuit and wirelength of interconnections. Therefore, floorplanning has a significant role in Printed Circuit Board (PCB) and Integrated Circuit (IC) design. The area covered by the circuit and the wirelength of the interconnections affect the cost of production [2].

Floorplanning problem is an NP hard problem from the computational point of view [1]. The size of the search space grows exponentially with the number of modules. Thus, it is a very challenging task to find an optimum solution. Speed and quality of the floorplanning process is directly related to the representation scheme of floorplan. The scope of search space and the complexity of transformation between the floorplanning representation and its corresponding floorplan are determined by the representation method. Researchers have proposed many representation methods such as Polish Notation, Bounded Sliced Grid (BSG), Transitive Closure Graph (TSG), B*Tree, and Sequence Pair.

In addition to the representation method, the floorplanning algorithm is another essential factor for speed and quality of the floorplanning process. There are different algorithms, which have been used by researchers, starting from Simulated Annealing (SA) to the Relay Race Algorithm (RRA) [3, 4]. Simulated Annealing is the first floorplanning algorithm, which was utilized by Wong and Liu to optimize the area of floorplan [5]. This metaheuristic algorithm is still used by many floorplanners today. Rebandengo and Reorda have used Genetic Algorithm (GA) as an evolutionary algorithm [6]. Sheng, Takahashi and Ueno have designed Relay-Race Algorithm to overcome the shortcomings of Simulated Annealing and Genetic Algorithm [4]. In this thesis, a Modified Relay-Race approach has been implemented to improve the solution quality and the speed of the process. Sequence Pair has been chosen as a representation scheme for floorplan.

## 1.1. Physical Design Process

To cope with the increasing complexity of electronic circuits, hierarchical design is the commonly used procedure in PCB and IC design [7]. The main stages of physical design process are depicted in Figure 1.1.

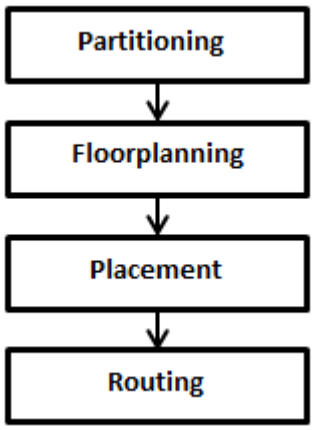



Figure 1.1. Physical Design Process.

Partitioning is the first stage of the physical design process. The circuit may consist of millions of transistors, which tremendously increases the complexity of the tasks in the following stages. Partitioning allows dealing with smaller subcircuits rather than the entire circuit at once, making the layout efforts more manageable.

Floorplanning is the crucial stage in the physical design process. In this stage, the relative layout placements for each module, as well as the entire circuit are decided.

In the placement stage, the transistors or gates within the modules are exactly located on the layout area. The aim of the placement stage is to reach a minimum area formation for the modules that allows completion of interconnections.

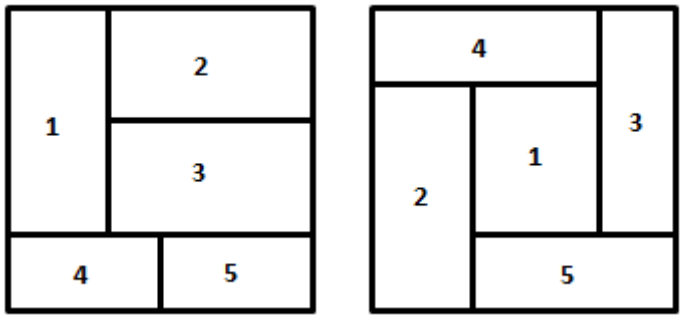
During the routing stage, interconnections between the modules are established according to the specified netlist. Different types of metal layers available for drawing the wires that establish these interconnections.

## 1.2. Floorplanning Problem

Floorplanning is the determination of module locations considering the criteria such as area and wirelength. A set of $n$ rectangular modules $M = \{m_1, m_2, m_3, \ldots, m_n\}$ is the input the to the floorplanning problem. Each module $m_i$, $1 \leq i \leq n$, has a rectangular shape with a height $h_i$ and a width $w_i$. A set of $k$ nets $N = \{n_1, n_2, n_3, \ldots, n_k\}$ is another input to the flooplanning. These nets are the interconnects between modules. Each net $n_i$, $1 \leq i \leq k$, has a length $l_i$. The pin locations of modules can also be input to the floorplanning. In short, the main inputs for the floorplanning are a module set $M = \{m_1, m_2, \ldots\}$ with the height and width $\{(h_1, w_1), (h_2, w_2), \ldots\}$ and a net set $N = \{n_1, n_2, \ldots\}$. The aim of floorplanning is to determine the locations of the modules in order to minimize the area of the rectangular bounding box and the total wirelength of interconnections while not allowing overlap of any two modules.

There are two types of modules in floorplanning: soft and hard modules [8]. Soft modules have a fixed area but the width and height can be varied such that aspect ratio given by $R_i = w_i/h_i$ remains the same. On the other hand, hard modules have a fixed area, width and height.

There are also two types of floorplans, which are called slicing floorplans and non-slicing floorplans [9]. Slicing floorplans can be bisected repetitively until each part includes only one module. This type of floorplans can be represented by a binary tree, called the Slicing Tree, having modules at the leaves and bisection types indicated by H for the horizontal bisection and V for the vertical bisection at the internal nodes. Figure 1.2(a) depicts an instance of a slicing floorplan. Non-Slicing floorplans are a more general form of floorplans. These floorplans cannot be bisected; therefore, they cannot be represented by Slicing Trees. Constraint Graph Pair (CGP) method is utilized in order to model these floorplans. CGP method consists of Horizontal Constraint Graph (HCG) and Vertical Constraint Graph (VCG). HCG and VCG defines the horizontal and vertical relations among the modules respectively. Figure 1.2(b) depicts an instance of a non-slicing floorplan.




(a) Slicing Floorplan (b) Non-Slicing Floorplan

Figure 1.2. Types of Floorplan

The main objective of floorplanning is to optimize a layout according to a predefined cost function [9]. The most common consideration in this function is the area covered by the rectangular bounding box enclosing all modules. This requires minimization of the dead space, which is called white space. White space is a space that is not covered by any module in the floorplan. The second most common consideration in this function is the total wirelength, which has several types of evaluations such as Minimum Chain Method, Steiner Tree Estimation and Half Perimeter Wirelength (HPWL). The HPWL is the most efficient method among these methods and it is given by half the perimeter of the bounding rectangular box that surrounds all the pins of the net to be connected [10]. A cost function that is widely utilized in floorplanning is a combination of area and wirelength as given by Equation 1.1.

$$Cost = \alpha.C_a + (1 - \alpha).C_w \tag{1.1}$$

In the Equation 1.1, $C_a$, $C_w$ and $\alpha$ represent area function, wirelength function and the weight factor respectively. The weight factor $\alpha$ is associated with each objective and is user defined.

## 1.3. Representation of Floorplan

Representation of a floorplan is the fundamental aspect for floorplanning. Complexity of the transformation and the scope of search space are directly related to the represenation of the floorplan [1]. Representation of the floorplan has to be selected through the type of floorplan. Researchers have proposed several representation schemes in the last couple decades. Polish Notation, Bounded Slicing Grid, Transitive Closure Graph, B*Tree and Sequence Pair are the most commonly used representation schemes [11]. In Table 1.1, the comparison of different floorplan representation schemes is represented. This comparison contains information about the flexibility and the computational complexity of these floorplan representation schemes.

Table 1.1. Comparison of different floorplan representations

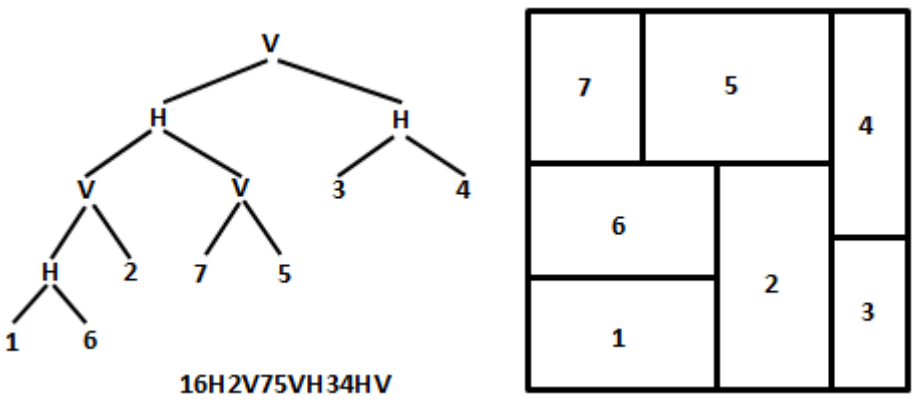
| **Representation** | **Flexibility** | **Complexity** |
|---|---|---|
| Polish Notation | Slicing | $O(n)$ |
| Bounded Slicing Grid | General | $O(n^2)$ |
| Transitive Closure Graph | General | $O(n^2)$ |
| B*Tree | Compacted | $O(n)$ |
| Sequence Pair | General | $O(n^2)$ |

As shown in Table 1.1, Polish Notation and B*Tree have better computational complexity and Bounded Slicing Grid, Transitive Closure Graph and Sequence Pair have better flexibility. Polish Notation is an efficient representation scheme for slicing floorplans, but it can not handle other types of floorplans. B*Tree is also an efficient representation scheme with smaller encoding cost. However, it is less flexible than Bounded Slicing Grid, Transitive Closure Graph and Sequence Pair. Bounded Slicing Grid is a flexible representation scheme, but it can not handle non-slicing floorplans. Transitive Closure Graph has faster run time and less memory usage. On the other hand, it can not handle slicing floorplans. Sequence Pair is very flexible representation scheme and it can handle all types of floorplans, but it has high encoding cost.

### 1.3.1. Polish Notation

Polish Notation representation is only suitable for Slicing Floorplans. A Binary Slicing Tree can be expressed using Polish Notation, $E = e_1 e_2 \dots e_{2n-1}$ where $e_1 \in \{1, 2, \dots n, H, V\}$. In this expression, operands (numbers) represent modules while H & V operators represent a horizontal and vertical cut respectively for a slicing floorplan. Expression of Polish Notation is the postfix ordering of a binary tree, which can be reached from the post-order traversal on a binary tree.

An important requirement for the Polish Notation is preservation of the balloting property. Balloting property dictates that the number of operands must be more than the number of operators for sequence $e_1$ to $e_i$ for all $i$. Balloting property must be observed when random Polish expressions are generated, because a Polish expression that does not obey the balloting property can not represent a valid slicing tree. In order to convert a normalized Polish expression to its corresponding floorplan, bottom-up procedure can be utilized to recursively combine the slicing sub-floorplans on the basis of Polish expression. Figure 1.3 illustrates an instance of Polish Notation and the corresponding floorplan.



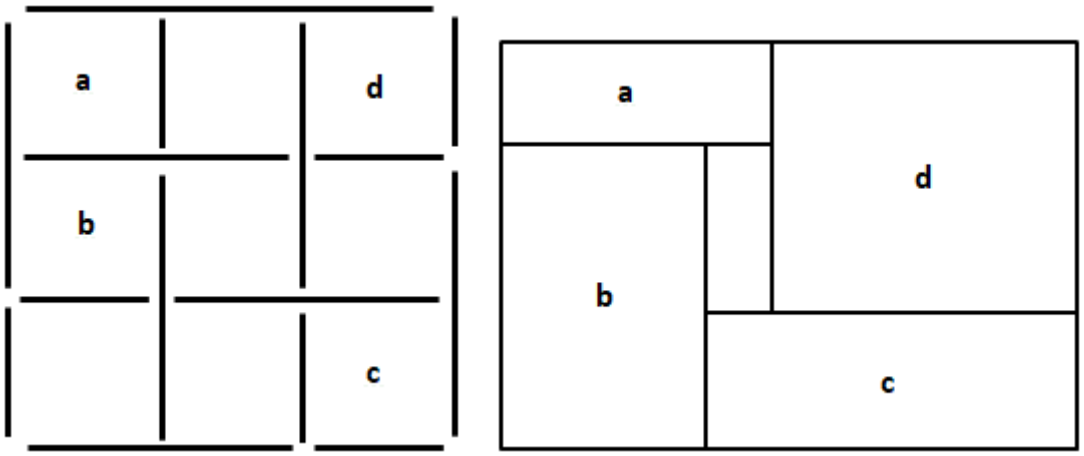

(a) Binary Tree and Its Polish Notation (b) Corresponding Floorplan

Figure 1.3. Polish Notation and Its Corresponding Floorplan

### 1.3.2. Bounded Slicing Grid

Bounded Slicing Grid representation is suitable for Non-Slicing Floorplans. It was proposed by Nakatate in 1996 [12]. In Bounded Slicing Grid, a row of non-overlapping horizontal line segments of two unit length is retrieved. Then, they are repeated for each row, considering a shift of one unit length between the adjacent rows. In other words, $n$ pieces of blocks are placed in special $n$ by $n$ grid. Proceeding in a similar way, a set of columns of vertical non-overlapping line segments of two unit length can be generated. These lines are called Bounded Slicing Lines. The rectangular region encased by these Bounded Slicing Lines is known as a room.

In Bounded Slicing Grid, floorplanning is maintained by task of blocks. If a room doesn't contain any blocks, this room is called an empty room. On the other hand, if a room contains one or more blocks, this room is called an occupied room. Figure 1.4(a) illustrates an instance of BSG. The corresponding floorplan of this instance is shown in Figure 1.4(b).



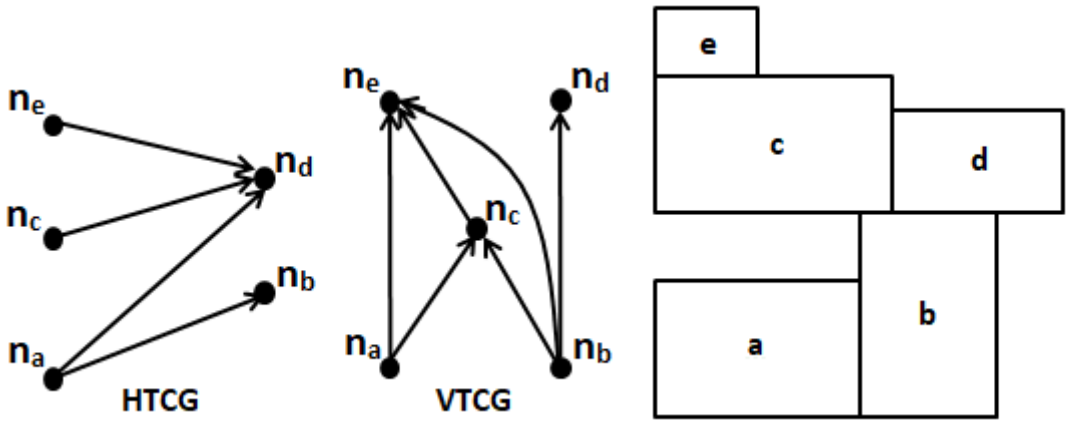

(a) BSG (b) Corresponding Floorplan

Figure 1.4. BSG and Its Corresponding Floorplan

### 1.3.3. Transitive Closure Graph

Transitive Closure Graph (TCG) is a directed acyclic graph $G = (V, E)$ [13]. TCG consists of two graphs as Horizontal Transitive Closure Graph (HTCG) and Vertical Transitive Closure Graph (VTCG). TCG representation defines the geometric relations between modules based on these two graphs. HTCG describes horizontal geometric relations between modules and VTCG describes vertical geometric relations among modules.

In TCG, geometric relations between modules and operations are transparent. Furthermore, TCG supports increasing updates during operations and keeps the information of boundary modules as well as the shapes and the relative locations of modules in representation. Figure 1.5 illustrates an instance of TCG representation and the corresponding floorplan. As Figure 1.5(a) indicates, TCG includes two graphs; namely, HTCG and VTCG. The corresponding floorplan of HTCG and VTCG is shown in Figure 1.5(b).



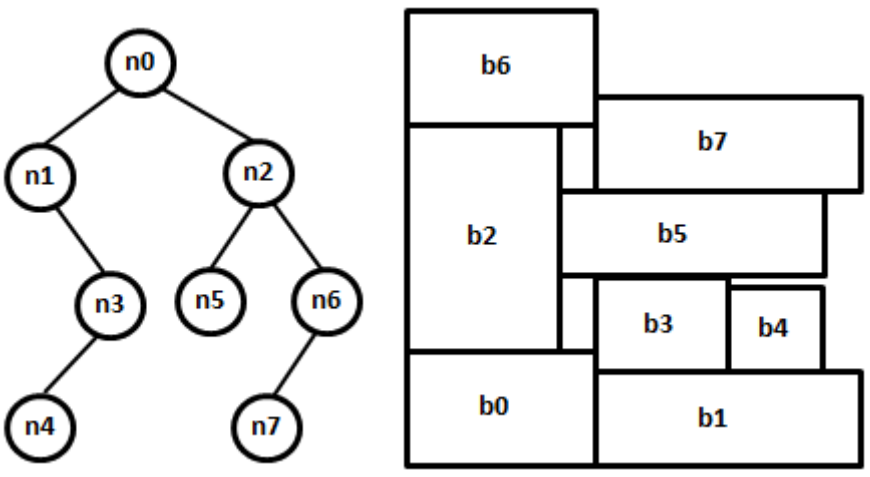

(a) TCG (b) Corresponding Floorplan

Figure 1.5. TCG and Its Corresponding Floorplan

### 1.3.4. B*Tree

B*Tree representation is based on ordered binary trees and used to model compacted floorplan structures. This representation scheme is suitable for Non-Slicing Floorplans. B*Tree can also be made by converting an O-Tree to binary tree. In a compacted floorplan structure, blocks cannot be shifted left or bottom. The corresponding block of the root of B*Tree locates on the bottom-left corner of the placement. The construction of a B*Tree starts from the root. Then, the left sub-tree and then the right sub-tree is recursively constructed. The left child of the node $n_i$ represents the lowest unvisited block in floorplan. The right child of $n_i$ corresponds to the lowest block located above and with its x-coordinate equal to that of $b_i$ and its y-coordinate less than the top boundary of the block on the left-hand side and adjacent to $b_i$.

All combinations of a B*Tree is obtained by perturbing the B*Tree using three types of movements. First movement is Node Movement which deletes a node or inserts a node in another location. Second movement is Node Swap which swaps two blocks. Third movement is Block Rotation which doesn't affect the tree structure. Figure 1.6 illustrates an instance of B*Tree and its corresponding floorplan.



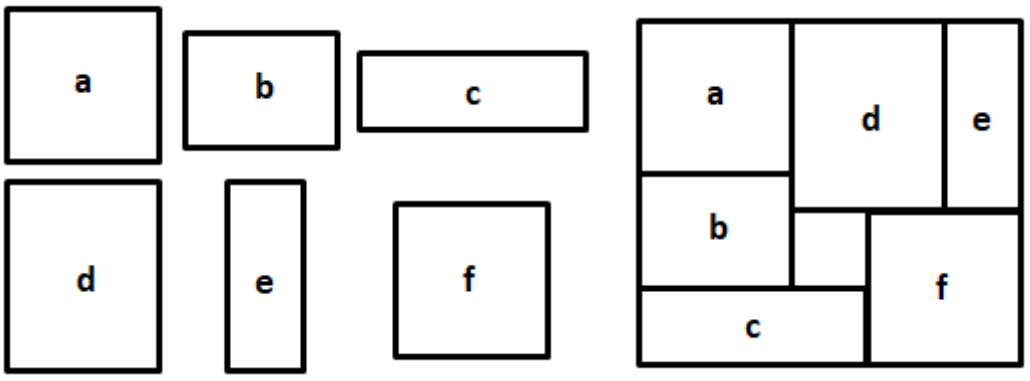

(a) B*Tree (b) Corresponding Floorplan

Figure 1.6. B*Tree and Its Corresponding Floorplan

### 1.3.5. Sequence Pair

Sequence Pair representation is the most flexible representation scheme. It is suitable for both Slicing and Non-Slicing Floorplans. A sequence pair $(\Gamma+, \Gamma-)$ for a set of $n$ modules is a pair of sequences of the $n$ module names [14]. A sequence pair imposes horizontal/vertical (H/V) constraints for every pair of modules. The oblique-grid notation of a given floorplan indicates H/V constraints specified by a given sequence pair. In order to construct a Sequence Pair, first of all, the Constraint Graph Pair i.e. the HCG and VCG has to be generated. For instance, (135642,634215) can represent $(\Gamma+, \Gamma-)$ as the sequence pair.

Given two modules $a$ and $b$, if $a$ comes after $b$ in $\Gamma+$, i.e., $< \ldots b \ldots a \cdots >$, and $a$ comes after $b$ in $\Gamma-$, i.e., $< \ldots b \ldots a \cdots >$, then $a$ has to be located to the right of $b$. If $a$ comes before $b$ in $\Gamma+$, i.e., $< \ldots a \ldots b \cdots >$, and $a$ comes before $b$ in $\Gamma-$, i.e., $< \ldots a \ldots b \cdots >$, then $a$ has to be located to the left of $b$. If $a$ comes after $b$ in $\Gamma+$, i.e., $< \ldots b \ldots a \cdots >$, and $a$ comes before $b$ in $\Gamma-$, i.e., $< \ldots a \ldots b \cdots >$, then $a$ has to be located below $b$. If $a$ comes before $b$ in $\Gamma+$, i.e., $< \ldots a \ldots b \cdots >$, and $a$ comes after $b$ in $\Gamma-$, i.e., $< \ldots b \ldots a \cdots >$, then $a$ has to be located above $b$. Figure 1.7(a) represents a set of modules and Figure 1.7(b) represents the corresponding floorplan of Sequence Pair for $(\Gamma+, \Gamma-) = (abdecf, cbfade)$.




(a) A Set of Modules (b) Corresponding Floorplan

Figure 1.7. Modules and floorplan of the SP $(\Gamma+, \Gamma-) = (abdecf, cbfade)$

# 2. EXISTING FLOORPLANNING APPROACHES

As already stated, floorplanning is determining the relative positions of modules while optimization goals like minimum area, minimum wirelength, or combination of both minimum area and minimum wirelength can be achieved. There are several heuristic and metaheuristic floorplanning algorithms that have been used with different representation methods. Linear Ordering and Cluster Growth Method are the first used algorithms in floorplanning [1]. These algorithms work on the principle of repetitive insertion of blocks along a direction.

Developments in the field of optimization led many researchers to utilize modern optimization methods. Simulated Annealing is the first modern optimization algorithm that has been used to optimize the floorplanning area. Then, researchers have begun to utilize the population-based metaheuristic algorithms, which imitate natural biological evolution and the social behavior of species. These algorithms have been collectively named as Evolutionary Algorithms. There are evolutionary algorithms that have been used to optimize the different types of cost functions with the various representation methods mentioned in the previous chapter. Genetic Algorithm, Ant Colony, Particle Swarm Optimization, and Differential Evolution are in the category of evolutionary algorithms [15–18]. Genetic Algorithm is the most commonly used evolutionary algorithm among these algorithms.

Despite the fact that various different ideas have been utilized to improve the search efficiency, the efficiency of existing search algorithms is still limited. It is expected that a better solution may exist near an obtained local optimum solution. However, potentially good solutions that are similar to the obtained local optimum solution are not effectively explored by existing search algorithms. Therefore, novel algorithms that are further improved are still being developed by researchers.

## 2.1. Simulated Annealing

Simulated Annealing is one of the earliest and most popular algorithms used in floorplanning optimization [3]. The implementation of Simulated Annealing is simple and it has been successfully utilized with different representation methods such as Polish Notation, Bounded Slicing Grid, B*Tree, and Sequence Pair [3, 5].

Simulated Annealing has been proposed based on the theory of statistical mechanics and the analogy between solid annealing and optimization problem [3]. The utilization of Simulated Annealing algorithm to solve the floorplanning problem was first introduced by Otten in 1983 [19]. Simulated Annealing resembles the cooling procedure of molten metal through annealing. In the cooling process of molten metal, the atoms have the highest mobility at high temperatures. As temperature drops, the movement ability of the atoms also reduce. The atoms then slowly begin to get organized and finally form crystals having the minimum possible energy state.

Each state of the solid structure corresponds to the applicable solution of problem. The energy of the state is the value of cost function to assess the solution. The state of the lowest energy represents the optimal solution with the best value of cost function. Simulated Annealing is a stochastic algorithm with iterative improvements. Each repetitive step includes an alteration of the current solution to a new solution. This action is named as movement to neighbourhood. Current temperature of the state determines the acceptance probability of new solutions. Temperature updates are scheduled from the highest temperature to the lowest temperature, where the acceptance probability at higher temperatures is higher than the acceptance probability at lower temperatures. If the temperature is decreased rapidly, it is known as Simulated Quenching, instead of Simulated Annealing. The difference between Simulated Annealing and Simulated Quenching is the parameter adjustment of temperature scheduling. In Simulated Annealing, the temperature needs to be decreased at a slower rate in order to reach the absolute minimum energy state.

Let $S$ be the solution space with neighbourhood structure on which Simulated Annealing searches to minimize the cost function $C(S_c)$ of current solution $S_c$. If no better solutions can be reached by traversing any neighbouring solution $S_n$, then the current solution $S_c$ represents a local optimum solution $S_o$. The inequality $C(S_c) < C(S_n)$ is always satisfied by the neighbour solution $S_n$ of local optimum solution $S_o$. The maximum value of $D(S_o)$ in the inequality $C(S_o)+D(S_o) > C(S_n)$ corresponds to the depth of local optimum solution. Furthermore, the notation $d(S_o)$ represents the maximum depth of local optimal solution $S_o$ in the solution space. Let $T_c$ and $X(T_c)$ correspond to the current temperature and the value of the cost function $C(S_c)$ respectively. Let $C_{min}$ represents the minimum value for the cost of the current solution. The conditions below satisfy the equality $\lim_{c\to\infty} X(T_c) = C_{min}$ [20]:

(i) Irreductible and finite solution space $S$,
(ii) The existence of the equilibration distribution of transition likelihood matrix
(iii) $T_c > 0$ and $T_c \geq T_{c+1}$ and for all $c$,
(iv) $\lim_{c\to\infty} X(T_c) = 0$,
(v) $\sum_{c=0}^{\infty} exp(-d(S_o)/T_c) = \infty$

In a practical implementation, the algorithm has to be simulated in finite computation time. Since the solution space of SA is connected, there is a search path from the initial solution to an optimum solution by a finite number of iterations. The diameter of the solution space is small; therefore, the chance of reaching an optimum solution by a small number of iterations is increased. The difference of costs between adjacent solutions is small; thus, the convergence and stability are improved. Consequently, a set of moving methods in SA is designed with the aim of having a small number of adjacent solutions and a small difference of costs between solutions. However, this type of moving methods can not cause a drastic change in results. This feature decreases the efficiency of search ability for the global optimum solution.

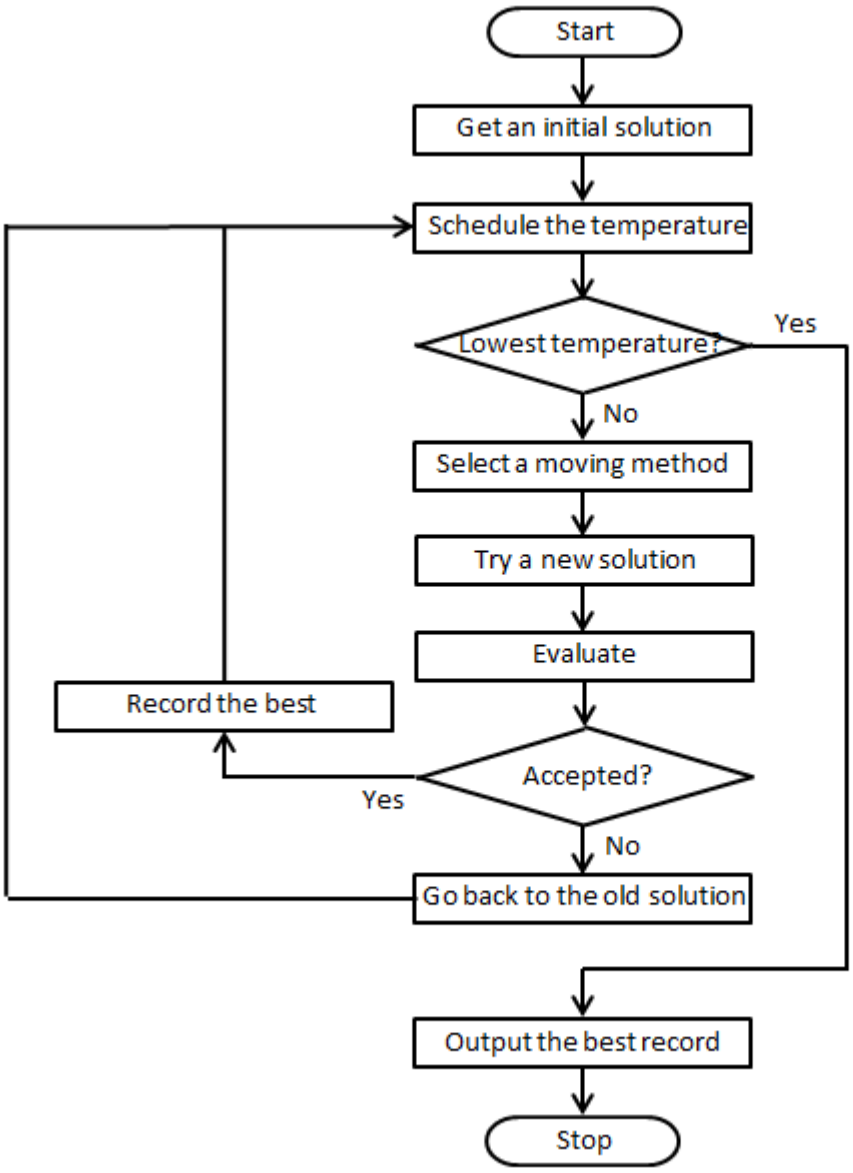



Figure 2.1. Flowchart of Simulated Annealing

Figure 2.1 represents a typical flowchart of Simulated Annealing. The initial solution of the algorithm can be user defined or randomly generated. If the algorithm is repetitively computed, the initial solution can be also the output of the past runs. The scheduling of the temperature is utilized to alter the current temperature $T$. The parameters of the temperature scheduling consist of the starting temperature $T_0$, the finishing temperature $T_f$, a temperature coefficient, and a loop count. A moving method has to be selected after the temperature scheduling. The selection of the moving method can be made randomly or according to given probabilities. For instance, the same probability for each moving method is near 33% in the case of three moving methods. Then, a new solution is tried by utilizing the current selected moving method. The new solution is evaluated by the cost function $C(\mathrm{S})$. After the evaluation of the new solution, the cost values of new and old solution are compared. The acceptance of the new solution depends on the calculated acceptance probability $P = exp(-\Delta C/T)$. The acceptance probability is calculated with using the difference of cost function $\Delta C$ and the recent temperature $T$. The value that this probability $P$ value can take is between 0 and 1. The temperature coefficient controls the speed of temperature reduction. The value that the temperature coefficient value can take is also between 0 and 1. The number of iterations for the inner loop is the number of repeated movements for each temperature $T$. If the new solution is better than the old solution ($\Delta C < 0$), then $P = 1$, and the new solution will be recorded. If the new solution is also better than the current best, the best solution will be replaced. If the new solution is worse than the old solution, it will be accepted with probability acceptance based on temperature; otherwise, the current solution will go back to the old solution. The algorithm will continue to the next temperature scheduled until the termination condition of SA is satisfied. The termination condition of SA is that the current temperature reaches the predetermined lowest temperature value. When the termination condition is satisfied, the output of the algorithm will be the best record. The practical implementation of Simulated Algorithm mainly depends on four definitions: solution representation, moving methods, cost function, and temperature scheduling.

## 2.2. Genetic Algorithm

Genetic algorithms have been widely investigated as a possible algorithm to solve many optimization and search problems in the last three decades [21]. Genetic algorithms belong to the class of evolutionary algorithms, based on a mechanism which mimics the way nature follows to improve the characteristics of living beings.

Genetic algorithms operate with a random population of individuals. Each individual of the population corresponds to a solution of the problem. These solutions have been represented as a chromosome of genes and each gene may have one of several values. Each solution has a fitness value that is calculated by an evaluation function. The fitness value of a solution indicates the distance between this solution and the optimum solution.

The population of individuals evolves from generation to generation through the creation of new individuals and the deletion of some individuals. According to the evolutionary theories, the most suited individuals of the population are likely to survive and generate new individuals. Therefore, biological heredity is transmitted to new individuals. The process begins with randomly generated population of individuals. Then, new individuals are generated through evolution, which is mainly based on two mechanisms: crossover and mutation.

Crossover is the first mechanism of the evolution procedure. In the crossover operation, two individuals from the current population are selected and their chromosomes are compounded with a certain rate to generate a new individual. In general, two individuals are used to generate two new individuals with two crossover operations. These new individuals are inserted into the current population. Then, the individuals are ranked with respect to their fitness values and some individuals with the lowest fitness values are eliminated from the population. Selection of individuals can be operated randomly or based on the probability of each individual proportional to its fitness value. This process is directed toward the regions of search space, where optimal solutions are supposed to exist.

Mutation is the second mechanism of the evolution procedure. In the mutation operation, a gene of a selected individual is randomly altered. In other words, the chromosome of the individual is changed by replacing a selected gene with new gene. Thus, mutated individuals gain new features. This operation allows to reach unexplored regions in search space.

Genetic Algorithm has been utilized as a floorplanning algorithm after Simulated Annealing by the researchers. Rebandego and Reorda are the first researchers who have used Genetic Algorithm to solve the floorplanning problem [6]. They have used Genetic Algorithm with Polish Notation in 1996. Afterwards, Nakaya *et al.* and Lin *et al.* have also presented Genetic Algorithms using Polish Notation for the floorplanning problem [22, 23]. Gwee and Lim have proposed Genetic Algorithm with heuristic based decoder for IC floorplaning in 1999 [24]. This approach was able to achieve an efficient solution to the multi-objective area and wirelength optimization problem of floorplanning. In 2006, Drakidis *et al.* and in 2007, Chatterjee and Manikas have presented Genetic Algorithm based floorplanning approach using sequence pair representation [25].

The utilization of Genetic Algorithm for floorplanning optimization starts with the randomly generated population of solutions. These solutions have random placement of modules along a defined rectangle of the circuit. The dimensions of modules in the circuit are also inputs to the algorithm. Then, the solutions are evaluated for their fitness values based on the predefined fitness function. The objective of this fitness function can be area or wirelength optimization or optimization for both criteria. The modules correspond to genes in chromosomes. After the creation of the initial population, the algorithm follows the mentioned mechanisms repetitively until the specified number of generations is reached. In the crossover operation, two floorplan solutions are taken and they are used to generate a new floorplan arrangement as a new solution. These new solutions are called as the offsprings of the selected solutions that the crossover operation is performed on. Afterwards, mutation operation is applied at lower rate by flipping any module of the solution. Finally, the new population is evaluated and some of the solutions with the lowest fitness value are deleted.

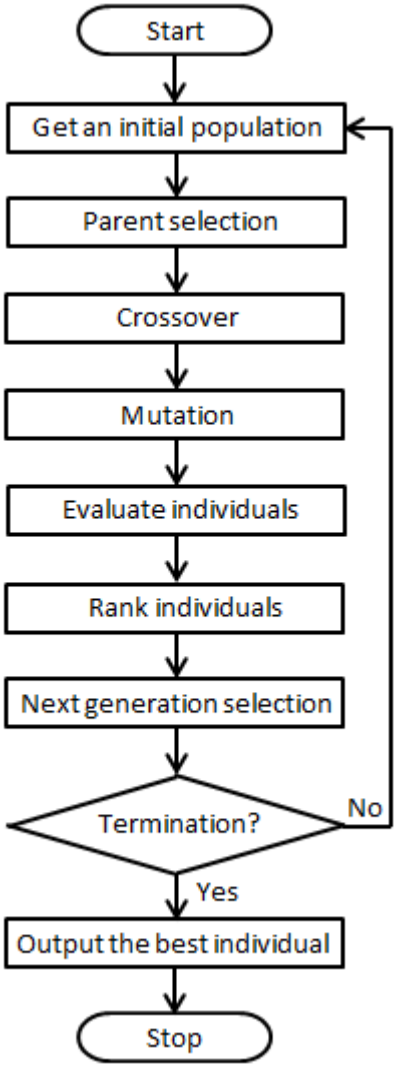



Figure 2.2. Flowchart of Genetic Algorithm

A typical flowchart of a Genetic Algorithm is illustrated in the Figure 2.2. The generation of initial population is the first step of the Genetic Algorithm. The number of solutions found in the population is important for the scope of search space. If the number solutions is very small, the desired solution diversity of evolutionary algorithm may not be achieved. On the other hand, the number of solutions also affects the computation time. If the number of solutions is too large, the computation time of the algorithm may exceed the desired upper limit of the computation time. Thus, the population size has to decided very carefully.

Parent selection prepares the population for the crossover operation. The selection of parent solutions can be made randomly in order to replicate the natural evolution procedure. The crossover is applied by the given crossover rate. The crossover rate determines the number of modules that will be replaced with modules from other parent solution.

Mutation is the second part of the evolution process. Mutation rate is also important for exploration of new regions in search space. If the mutation rate is very small, the algorithm may not reach the desired unexplored regions. If the mutation rate is too large, then the desired heredity can be lost.

After the evolution process is completed, each solution of the population is evaluated by the fitness function. The aim of the fitness function can be the improvement of area, wirelength or both. Then, the solutions in the population are ranked with respect to their fitness values. The selection of the next generation is made according to ranking of the solutions. Therefore, the fittest solutions survive to the next generation.

With the formation of the next generation, a cycle of the evolutionary algorithm is completed. Next cycles will follow the evolution procedure in the same order. The algorithm ends when the desired number of cycles is reached.

## 2.3. Relay Race Algorithm

Sheng *et al.* proposed the Relay Race Algorithm to approach a global optimal solution by exploring similar local optimal solutions more efficiently within shorter computation time for floorplanning problem [4]. Sheng *et al.* stated that Relay Race Algorithm was designed to overcome the shortcomings of SA and GA. Sequence Pair is used as a representation method.

Floorplanning problem has a finite solution space. Heuristic approaches are typically utilized to get near optimal solutions. SA and GA are two of the most popular stochastic algorithms for the floorplanning problem. However, there are some shortcomings of these heuristic approaches. The following shortcomings of SA can be improved:

(i) The experience of past movements is not used. This might lead to many unnecessary movements.
(ii) The global search is maintained by running away from the local optimum step by step. It may take excessive time to escape from the local optimum.

Furthermore, GA has also shortcomings as follows:

(i) The selection of next generation is made based on ranking function. The ranking of the population may take too much computational time.
(ii) The new solutions are generated using parent solutions. This might decrease the chance of potential solutions to join the competition if their parents are not good enough.

RRA has been designed in order to overcome the mentioned shortcomings of SA and GA. Relay Race Algorithm imitates the relay race competition where each runner runs a part of the race, and then is replaced (i.e. relayed) by another runner in the team.

The similarities between RRA and these two heuristic algorithms are as follows:

(i) RRA is also an iterative optimization algorithm.
(ii) These three algorithms belong to the stochastic method.

The differences between RRA and these two heuristic search approaches are as follows:

(i) RRA utilizes the experiences of past movements.
(ii) Escaping from local optimum can be made in one step by RRA.
(iii) The potential solutions have more chance to join the competition in the RRA.

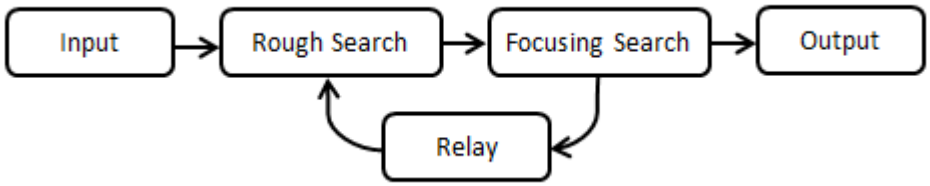
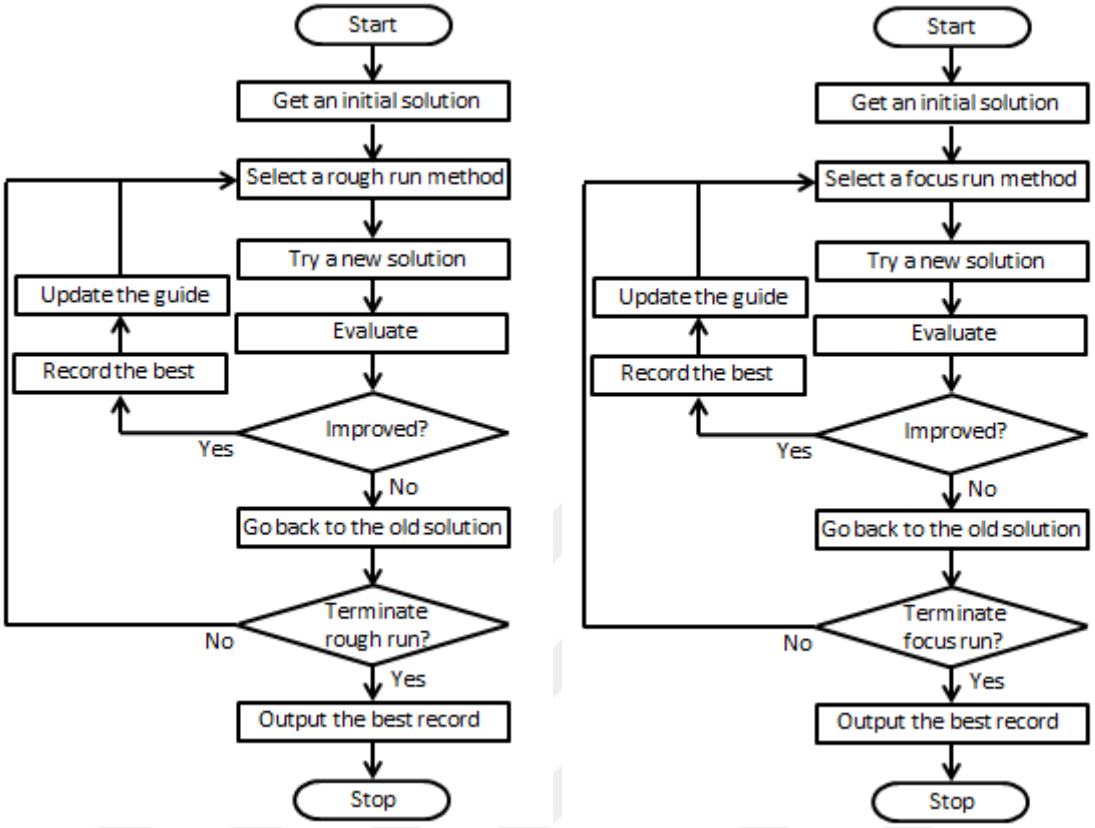



Figure 2.3. Basic Concept of Relay Race Algorithm

RRA contains the three basic parts shown in Figure 2.3: focusing search, rough search, and relay. The aim of the rough search is to pass over little hills in search space and approach a local optimum as quickly as possible. The focusing search tries to reach as close to the local optimum as possible. In focusing search and rough search, the current solution is repeatedly modified if there is an improvement. If the movement applied solution is worse than the previous solution, the movement applied solution is rejected and the previous solution is returned. The relay works for both running away from the local optimum with a single operation and maintaining the search continuity. Although the relayed solution is worse than the previous solution, the relayed solution is always accepted. The relayed solution should be both distant enough from the previous local optimum to run away from it and close enough to approach another local optimum.




(a) Rough search (b) Focusing search

Figure 2.4. Rough search and focusing search

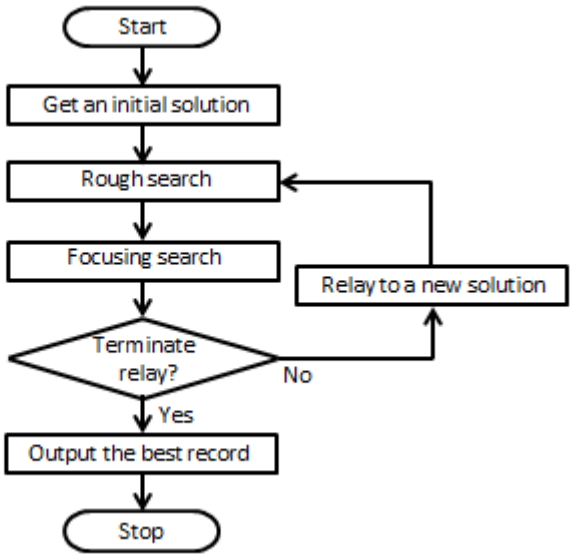



Figure 2.5. Flowchart of Relay Race Algorithm

The flowchart of RRA is shown in Figure 2.5. The initial solution of the RRA can be user defined or randomly generated. Figure 2.5 shows that the flowchart of RRA includes two inner loops. The first inner loop corresponds to rough search and second inner loop corresponds to focusing search. The rough search and focusing search can be named as rough run and focusing run due to the relay race concept.

Rough search begins with method selection. Three types of moving methods are utilized in rough search: group rotation, group exchange and group insertion. The group rotation rotates the randomly selected modules. The group exchange is the exchange of randomly selected modules. In the group insertion, the order of randomly selected modules in one sequence is changed. The number of modules in the group can be vary based on the total number of modules in the circuit. After the rough moving method is selected, rough search modifies the current solution to generate the next solution by utilizing the selected moving method. Then, the new solution is evaluated and if there is an improvement, the new solution becomes the current solution and best record and the guide are updated. Otherwise, the new solution is rejected and the old solution is kept as current solution. When the number of trials with no improvement reaches the predetermined number $N_r$, rough search is terminated.

Focusing search starts with the termination of rough search. After the rough search is completed, the local optimal solution is transmitted to focusing search. For the focusing search, three focusing moving methods are utilized: rotation, exchange and insertion. The rotation alters the orientation of a single module. The exchange is the exchange of the order of two modules in both sequence $\Gamma+$ and $\Gamma-$. In the insertion, the order of a single module is changed in one sequence. After the focusing moving method is selected, focusing search modifies the current solution to generate next solution by utilizing the selected moving method. Then, the new solution is evaluated and if there is an improvement, the new solution becomes the current solution and best record and the guide are updated. Otherwise, the new solution is rejected and the old solution is kept as current solution. When the number of consecutive trials with no improvement reaches the predetermined number $N_f$, focusing search is terminated.

Relay is the last part of the outer loop. After completion of rough search and focusing search, relay operator takes the current solution. Then, new solution is generated by relay operation. This new solution consists of two parts. The first part of the new solution is inherited from current solution. The second part of the new solution is randomly generated. The percentage of the randomly generated part of the solution is defined as the parameter $R_e$. The selection of the randomly selected part is also made randomly.

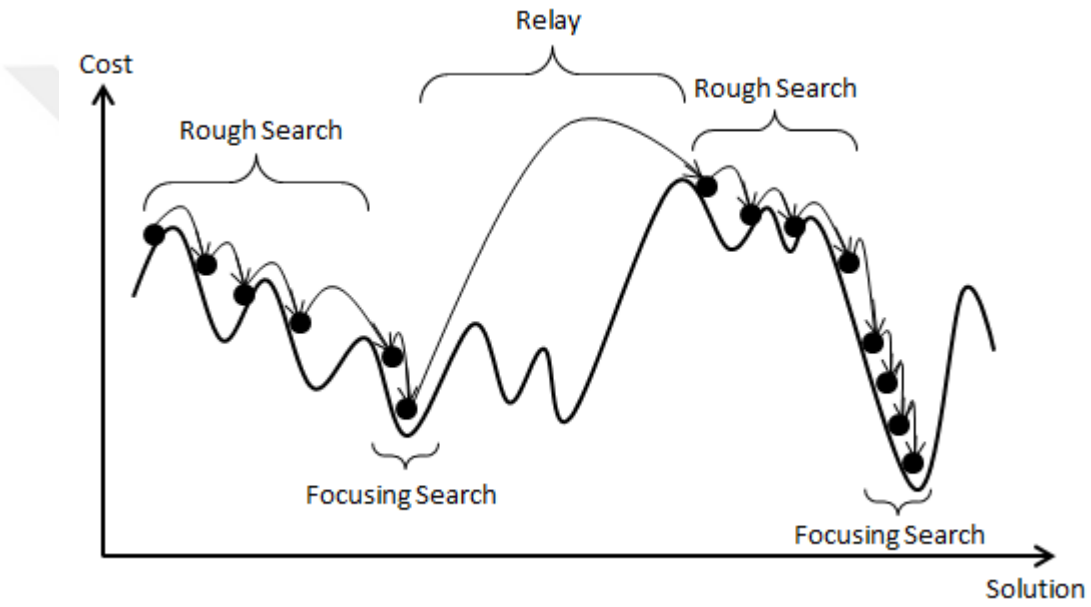



Figure 2.6. Behavior of Relay Race Algorithm in the Solution Space

Figure 2.6 illustrates the behavior of the RRA in the solution space. A solution is always improved in rough run and focusing run, while the solutions with no improvements are rejected. The differences between rough run and focusing run are in moving methods and in terminal condition. As the Figure 2.6 indicates, rough run gets over small hills and focusing search gets a local optimal solution. On the other hand, relay escapes from local optimum solution and reaches near another local optimum solution. This process is repeated as many times as the number of runners on the team $N_t$ to find the global optimum solution.

# 3. MODIFIED RELAY RACE APPROACH

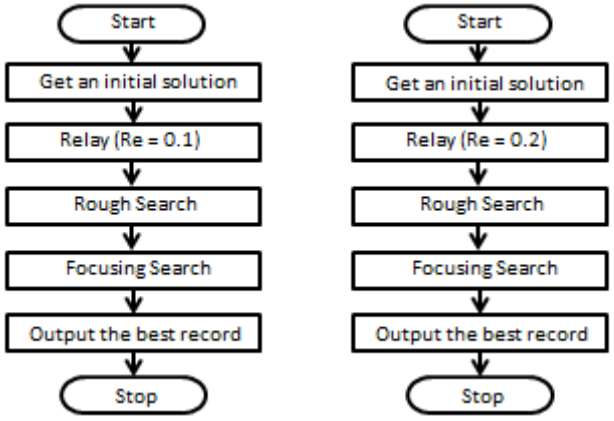
## 3.1. Overview

Although RRA has been proposed to overcome shortcomings of SA and GA, there are also shortcomings of RRA. A Modified Relay Race Approach (MRRA) is proposed in this thesis to improve the shortcomings of RRA. Sequence Pair scheme is utilized as a representation method with MRRA.

The main objective of the RRA is to achieve better solutions than the local optimum solutions achieved by the SA and GA. RRA implements three main operations in order to achieve this improvement: rough search, focusing search, and relay. The aim of rough search and focusing search is to obtain local optimum solution. On the other hand, the goal of the relay is escaping from the local optimum solution towards another unexplored local optimum solution. However, RRA has the following shortcomings which can be improved:

(i) The search is maintained by following a single path in the RRA. Searching with a single path may prevent the algorithm from exploring better local optimum solutions in solution space.
(ii) The parameters $N_r$ and $N_f$ determine the efficiencies of rough and focusing searches respectively. Defining these parameters as fixed numbers in the RRA may reduce the efficiency of the algorithm.

MRRA has been developed in order to overcome these shortcomings of RRA. In the MRRA, dual searching path method is applied to increase the chance of exploring better local optimum solutions. Moreover, maximum number of trials without improvement during rough search $N_r$ and maximum number of consecutive trials without improvement during focusing search $N_f$ are determined according to the detailed analysis, which also considers the number of modules in the circuit to improve the efficiency.




(a) Search path ($R_e = 0.1$) (b) Search path ($R_e = 0.2$)

Figure 3.1. Dual path search

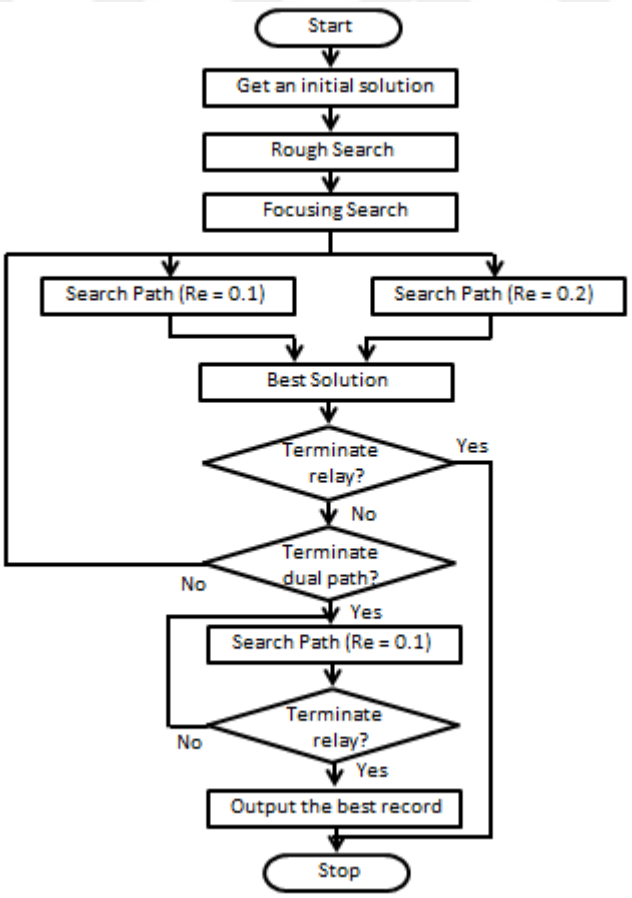



Figure 3.2. Flowchart of Modified Relay Race Algorithm

Figure 3.2 illustrates the flowchart of MRRA, which includes two inner loops. While the first inner loop corresponds to the dual path search of MRRA, the second inner loop corresponds to the single path search of the original Relay Race Algorithm.

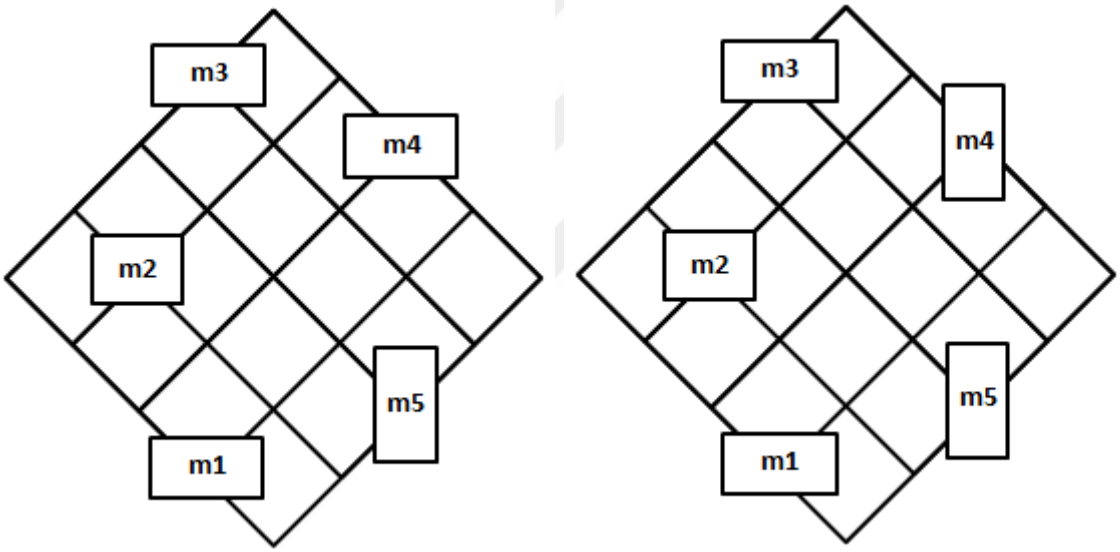
MRRA starts by getting an initial solution which can be user defined or randomly produced. Then, rough search and focusing search are applied to the initial solution. Afterwards, current solution enters the dual path search.

During dual path search, both paths implement the same operations. The only difference between these two paths is the percentage of randomly generated part of the solution $R_e$ in the relay operation. The implementation of searching process with two different paths increases the likelihood of achieving a better local optimal solution as the next solution. The value of the parameter $R_e$ used in the first path process is chosen to be 0.1, as in the original algorithm. The value of the parameter $R_e$ used in the second path process is chosen to be 0.2, which is larger than the $R_e$ value used in the first path; therefore, the second path makes it possible to search in farther regions of the solution space. Hence $R_e$ corresponds to the mutation rate as mentioned in the previous chapter, the current solution is mutated at a rate of 0.1 and 0.2 in the first path and in the second path respectively. Thus, there is an increased chance for exploring better local optimum solutions. After both first and second paths complete their search operations, the best solutions of these paths are compared, and only the better solution is kept as the next solution. However, if dual path search is applied until the algorithm is terminated, the computation time will increase. For this reason, there is also a termination condition for dual path search. When two consecutive solutions of the first path ($R_e$=0.1) are better than the solutions of the second path ($R_e$=0.2), dual path search is terminated, and the algorithm continues with single path search.

Single path search phase continues until the total number of runners in the team for the relay $N_t$ is reached. In the single path search, the value of the parameter $R_e$ is chosen to be 0.1. As a result of the dual path search, the probability of exploring better local optimum solutions in distant regions is increased.

## 3.2. Moving Methods

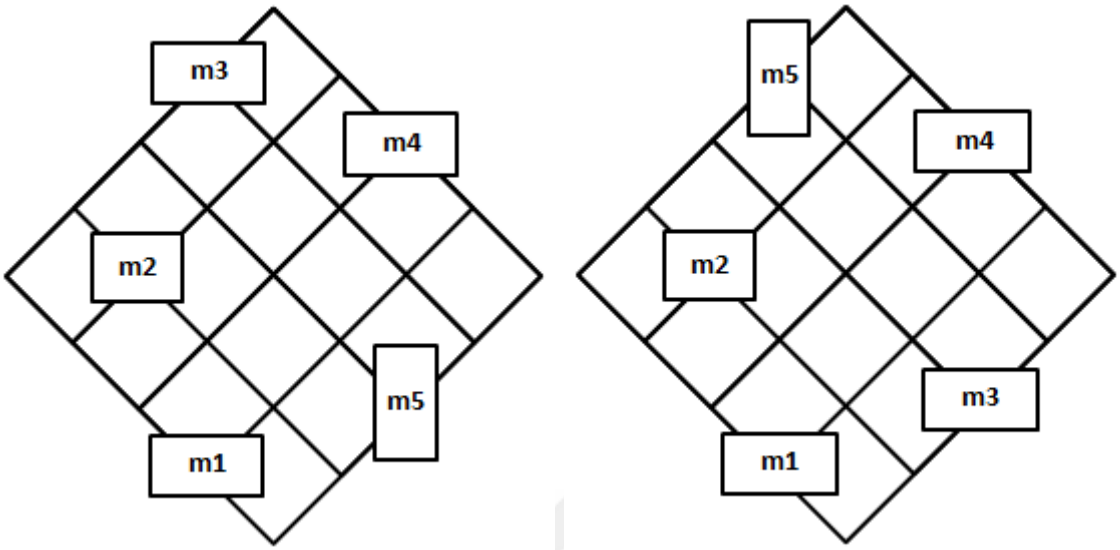
In order to solve floorplanning problem by MRRA, three focusing moving methods and three rough moving methods are utilized. Focusing moving methods are rotation, exchange, and insertion. Rough moving methods are group rotation, group exchange, and group insertion. These moving methods are exactly the same with the moving methods that are used in RRA. The aim of keeping the methods same with the original algorithm is to compare the differences in the results of RRA and MRRA which are only caused by modifying the approach.




(a) Before rotation (b) After rotation

Figure 3.3. Rotation Move

Figure 3.3 illustrates the utilization of the rotation move. The corresponding placement of the SP $(\Gamma+, \Gamma-) = (32415, 12534)$ is shown in Figure 3.3(a). This placement indicates the initial placement of modules before the rotation method is applied. Figure 3.3(b) illustrates the placement of modules after a rotation move is applied to module $m_4$. As seen in Figure 3.3(b), rotation move changes the orientation of some of the modules. If a rotation move is implemented for module $m_i$, orientation of module $r_i$ is replaced with $1 - r_i$, where $r_i \in \{0, 1\}$. The group rotation move is the implementation of rotation move for a set of randomly selected modules.

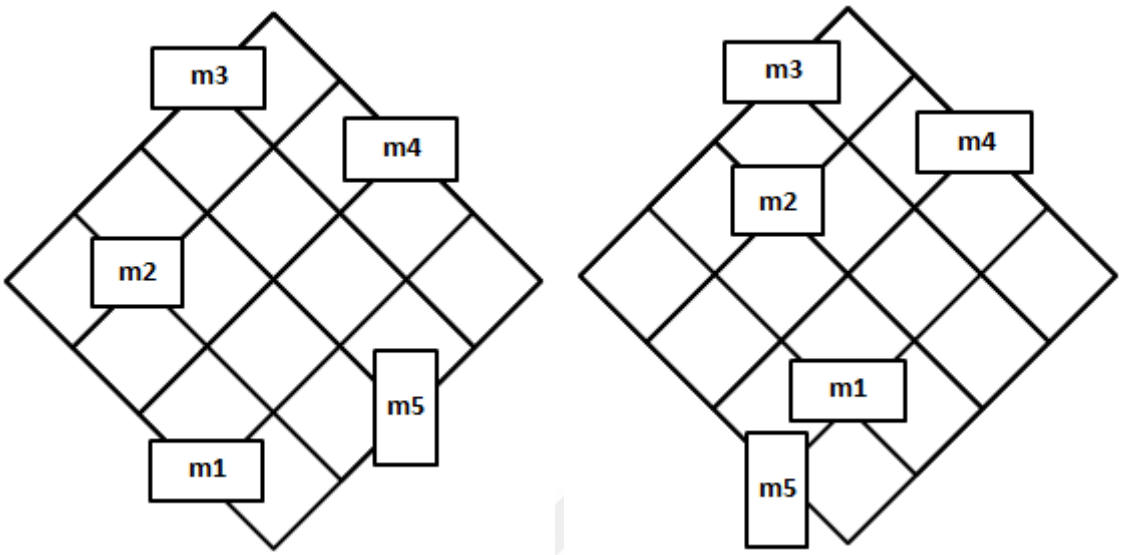



(a) Before exchange (b) After exchange

Figure 3.4. Exchange Move

Figure 3.4 illustrates the placement of modules before and after the implementation of the exchange move. Figure 3.4(a) illustrates the corresponding placement of the SP $(\Gamma+, \Gamma-) = (32415, 12534)$. This placement indicates the initial placement of modules before the exchange move is implemented. The placement of modules after the exchange move is implemented for the modules $m_3$ and $m_5$ is shown in Figure 3.4(b).

The exchange method exchanges the order of two modules both in positive sequence $\Gamma+$ and negative sequence $\Gamma-$. If an exchange method is applied to module $m_i$ and module $m_j$, $f_+(m_i)$, $f_-(m_i)$, $f_+(m_j)$ and $f_-(m_j)$ are altered to $f_+(m_j)$, $f_-(m_j)$, $f_+(m_i)$ and $f_-(m_i)$ respectively. For this instance, the exchange method is applied to module $m_3$ and module $m_5$. Therefore, $f_+(m_3) = 1$, $f_-(m_3) = 4$, $f_+(m_5) = 5$ and $f_+(m_5) = 3$ are changed to $f_+(m_3) = 5$, $f_-(m_3) = 3$, $f_+(m_5) = 1$ and $f_+(m_5) = 4$. The corresponding SP after the exchange method is implemented for the module $m_3$ and $m_5$ is $(\Gamma+, \Gamma-) = (52413, 12354)$. The group exchange move is the implementation of exchange move for a set of randomly selected modules.




(a) Before insertion (b) After insertion

Figure 3.5. Insertion Move

Figure 3.5 illustrates the utilization of the insertion move. Figure 3.4(a) depicts the corresponding placement of the SP $(\Gamma+, \Gamma-) = (32415, 12534)$. This placement is the initial placement of modules before the insertion move is applied. Figure 3.5(b) represents the placement of modules after the insertion move is implemented for the module $m_5$.

The insertion move changes the order of modules in one sequence. The selection of the sequence can be made randomly or user defined. If an insertion move is applied to module $m_i$ in positive sequence $\Gamma+$, $f_+(m_i)$ is changed to another value $f_+(m_j)$ and the orders of modules whose order is between $f_+(m_i)$ and $f_+(m_j)$ are shifted respectively. For this instance, the insertion move is applied to module $m_5$ in negative sequence $\Gamma-$. The orders of module $m_1$ and module $m_2$ are shifted accordingly. Therefore, $f_-(m_1) = 1$, $f_-(m_2) = 2$ and $f_-(m_5) = 3$ are changed to $f_-(m_1) = 2$, $f_-(m_2) = 3$ and $f_-(m_5) = 1$. The corresponding SP after the insertion move is implemented for the module $m_5$ is $(\Gamma+, \Gamma-) = (32415, 51234)$. The group insertion move is the implementation of insertion move for the randomly selected modules.

The selection of the moving method is implemented by using the probability of moving methods [4]. The probability of any method $p_{k+1}(i)$ is evaluated according to the old probability of the method $p_k(i)$ and the short term improvement speed $s_k(i) = a_k(i).f_k(i)/\sum_{j=1}^{3} a_k(j).f_k(j)$, where $i$ represents different moving methods, $a_k(i)$ is the relative amplitude of improvement and $f_k(i)$ is the frequency of the improvement. In detail, $a_k(i)$ is the relative $-\Delta C$ on average and $f_k(i)$ is the ratio of improved trials in the last $t$ trials. For rough moving methods and focusing moving methods, $t$ is chosen 30 and 100 respectively. If the evaluation of the solution satisfies the condition $-\Delta C > 0$, $a_k(i)$ is calculated and updated. Otherwise, the probability of methods are not updated. The new probability $p_{k+1}(i)$ is given by $p'_{k+1}(i)/\sum_{j=1}^{3} p'_{k+1}(j)$ for each moving method to keep the total probability 100%, where $p'_{k+1}(i)$ equals to $(p_k(i) + s_k(i))/2$.

For instance, for the following values for three moving methods:

- $[p_k(1), p_k(2), p_k(3)] = [50\%, 30\%, 20\%]$,
- $[a_k(1), a_k(2), a_k(3)] = [10\%, 30\%, 60\%]$,
- $[f_k(1), f_k(2), f_k(3)] = [90\%, 10\%, 50\%]$,

the new probability values for three moving methods will be as follows:

- $[p_{k+1}(1), p_{k+1}(2), p_{k+1}(3)] = [36\%, 18\%, 46\%]$.

The selection probability of moving methods depends on the performances of these moving methods in the last $t$ trials. If the relative improvement and frequency of the improvement of a moving method are larger than the relative improvement and frequency of the improvement of other moving methods, then the probability of this moving method for the selection of next moving method is relatively larger than the probability of other moving methods. The probability of moving methods with less efficiency are decreased in the search process. The utilization of the probability of moving methods allows benefiting from the experiences of past moves.

### 3.3. Cost Function

The cost function of MRRA has three objectives, including area, wirelength and overlap. Area is the size of the smallest rectangular bounding box that contains all modules. Wirelength is an approximation to the sum of all interconnects between modules. Overlap is the amount of overlap between the paths of same and different interconnects. The area and wirelength are the most common objectives for typical cost function in floorplanning. Overlap objective is inserted to cost function to consider interference between same and different types of nets. The cost function of MRRA is given by Equation 3.1.

$$C_t = \alpha.C_a + \beta.C_w + \gamma.C_o \tag{3.1}$$

In Equation 3.1, $\alpha, \beta, \gamma$ are user defined coefficients of their corresponding objectives that satisfy $\alpha + \beta + \gamma = 1$. $C_t$, $C_a$, $C_w$ and $C_o$ represent total cost function, cost function of area, cost function of wirelength and cost function of overlap respectively. Cost function of area $C_a$ estimates the area of bounding rectangular shape is given by the minimum bounding rectangle which includes all modules. The area is calculated with multiplication of the total width $W$ and the total height $H$. Cost function of wirelength $C_w$ estimates the total wirelength which is used for the connection of pins in circuit. Half perimeter wirelength method is utilized to obtain approximation of the wirelength for each net. Half perimeter wirelength is half perimeter of the minimum bounding box for all centers of module $m_i$ which has height $h_i$ and width $w_i$. For the net $n_i$, HPWL$[n_i]$ can be obtained from $(x_i, y_i, h_i, w_i)$. Cost function of overlap function $C_o$ estimates the total overlap cost between nets. The calculation of the overlap cost is made according to overlap coefficients. These overlap coefficients can be defined for both same and different types of nets (analog, digital, clock, etc) by the user.

# 4. EXPERIMENTS AND RESULTS

## 4.1. Parameter Setting

Determination of the parameters directly affects the performance and speed of the algorithm. Therefore, the best empirical values of parameters $N_f$, $N_r$, and $R_e$ are investigated by detailed experiments of floorplanning using ami33 which belongs to Microelectronics Center of North Carolina (MCNC) benchmark. MCNC benchmark is the most commonly used benchmark for the floorplanning optimization; therefore, it is also used in this thesis for comparison with other approaches. MCNC benchmark suite consists of five circuits: apte, xerox, hp, ami33 and ami49. The details of MCNC benchmark suite are shown in Table 4.1.

Table 4.1. The details of MCNC benchmark circuits

| **Circuit** | **Modules** | **Nets** | **Terminals** |
|---|---|---|---|
| apte | 9 | 97 | 73 |
| xerox | 10 | 203 | 2 |
| hp | 11 | 83 | 45 |
| ami33 | 33 | 123 | 42 |
| ami49 | 49 | 408 | 22 |

In the RRA, the best empirical values of parameters $N_f$, $N_r$ and $R_e$ are also investigated. However, this investigation does not contain intermediate values. For this reason, new investigation with intermediate values was implemented to obtain better values. The investigation for better parameter values also aims at speeding up the algorithm to remove the time increase due to the dual path search structure used in the MRRA. The values of the parameters were tested according to the number of modules in the circuit $N_m$ in order to be adaptive to the circuit. For the parameter setting, $\alpha$ is set to 1 and $\beta + \gamma$ is set to 0 in order to compare area optimization with published results.

Table 4.2 contains the results of RRA with parameters $N_r = 100$, $N_f = 1000$ and $N_t = 10$. The parameter setting was made by considering these results. In particular, the final cost, run time, and product of these values were used as the most important values in determining the parameters. In the tables, all units of initial and final costs are $(mm^2)$ and all units of run times are $(s)$. For the trial experiments, a set of initial solutions was generated. This set of initial solutions contains 100 initial solutions. Each trial experiment was conducted by utilizing this set of initial solutions to provide same initial conditions.

Table 4.2. Results of RRA $N_r = 100$ & $N_f = 1000$ & $N_t = 10$

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.334 | 1.805 | 2.409 |
| Best Cost | 3.353 | 1.259 | 1.904 | 2.397 |
| Best Time | 4.080 | 1.333 | 1.398 | 1.864 |
| Worst Cost | 3.885 | 1.396 | 1.803 | 2.518 |
| Median | 3.969 | 1.338 | 1.799 | 2.408 |

Table 4.3, Table 4.4, Table 4.5, Table 4.6, Table 4.7 and Table 4.8 contain the results of the investigation for the best combination of parameters $N_f$ and $N_t$. For this experiments, the value of $N_r$ has remained unchanged and is chosen to be three times the $N_m$ value, which equals the same value used in RRA. The parameter combination of $N_f = 10.N_r$ and $N_t = 10$ has better average final cost but its average run time is approximately 50% higher than the average run time of RRA as shown in Table 4.3. As the Table 4.4 and Table 4.5 indicate $(N_f, N_t) = (5.N_r, 10)$ and $(N_f, N_t) = (5.N_r, 15)$ have also better average final cost but their average run times are also approximately much more than the average run time of RRA. In Table 4.6, Table 4.7 and Table 4.8, the results of $(N_f, N_t) = (3.N_r, 15)$, $(N_f, N_t) = (3.N_r, 20)$ and $(N_f, N_t) = (3.N_r, 25)$ are shown respectively. The result of $(N_f, N_t) = (3.N_r, 20)$ is the only result which has better average final cost and better average run time simultaneously. These six trial experiments show that decreasing the $N_f$ while increasing the $N_t$ satisfies better results for MRRA.

Table 4.3. Trial experiments for $N_f = 10.N_r$ & $N_t = 10$ ($N_r = 3.N_m$)

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.303 | 2.765 | 3.603 |
| Best Cost | 4.611 | 1.246 | 2.514 | 3.133 |
| Best Time | 3.632 | 1.349 | 1.638 | 2.210 |
| Worst Cost | 3.800 | 1.385 | 2.468 | 3.419 |
| Median | 3.969 | 1.302 | 2.682 | 3.493 |

Table 4.4. Trial experiments for $N_f = 5.N_r$ & $N_t = 15$ ($N_r = 3.N_m$)

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.306 | 2.185 | 2.855 |
| Best Cost | 4.234 | 1.260 | 2.015 | 2.540 |
| Best Time | 3.965 | 1.319 | 1.665 | 2.197 |
| Worst Cost | 4.521 | 1.374 | 2.067 | 2.841 |
| Median | 3.969 | 1.303 | 2.127 | 2.772 |

Table 4.5. Trial experiments for $N_f = 5.N_r$ & $N_t = 20$ ($N_r = 3.N_m$)

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.304 | 2.793 | 3.643 |
| Best Cost | 3.935 | 1.246 | 2.326 | 2.898 |
| Best Time | 3.092 | 1.342 | 2.140 | 2.872 |
| Worst Cost | 3.546 | 1.378 | 2.597 | 3.581 |
| Median | 3.969 | 1.301 | 2.792 | 3.635 |

Table 4.6. Trial experiments for $N_f = 3.N_r$ & $N_t = 15$ ($N_r = 3.N_m$)

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.326 | 1.407 | 1.867 |
| Best Cost | 3.965 | 1.257 | 1.483 | 1.865 |
| Best Time | 2.970 | 1.334 | 1.054 | 1.406 |
| Worst Cost | 3.935 | 1.397 | 1.226 | 1.713 |
| Median | 3.969 | 1.325 | 1.373 | 1.820 |

Table 4.7. Trial experiments for $N_f = 3.N_r$ & $N_t = 20$ ($N_r = 3.N_m$)

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.307 | 1.744 | 2.281 |
| Best Cost | 3.700 | 1.249 | 1.585 | 1.980 |
| Best Time | 3.485 | 1.317 | 1.387 | 1.828 |
| Worst Cost | 4.987 | 1.392 | 1.538 | 2.142 |
| Median | 3.969 | 1.307 | 1.699 | 2.221 |

Table 4.8. Trial experiments for $N_f = 3.N_r$ & $N_t = 25$ ($N_r = 3.N_m$)

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.303 | 2.121 | 2.765 |
| Best Cost | 5.092 | 1.240 | 2.019 | 2.504 |
| Best Time | 2.296 | 1.293 | 1.691 | 2.188 |
| Worst Cost | 4.981 | 1.368 | 2.010 | 2.751 |
| Median | 3.969 | 1.301 | 2.053 | 2.672 |

Figure 4.1 represents the comparison in the average cost of parameter combinations in Table 4.3, Table 4.4, Table 4.5, Table 4.6, Table 4.7, and Table 4.8. As Figure 4.1 indicates the parameter combination in Table 4.3 and the parameter combination in Table 4.8 have the best average cost and the parameter combination in Table 4.3 has the worst average cost.

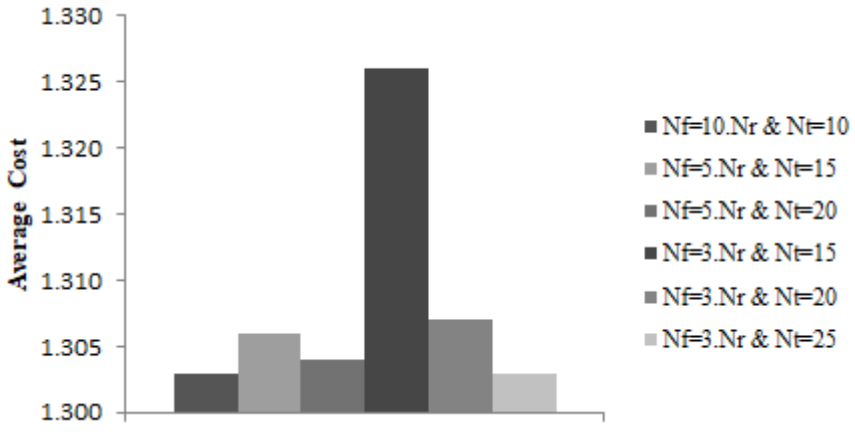



Figure 4.1. Average cost investigation of combinations $N_f$ & $N_t$

Figure 4.2 represents the comparison in the average run time of parameter combinations in Table 4.3, Table 4.4, Table 4.5, Table 4.6, Table 4.7, and Table 4.8. As Figure 4.2 indicates the parameter combination in Table 4.6 has the best average run time and the parameter combination in Table 4.5 has the worst average run time.

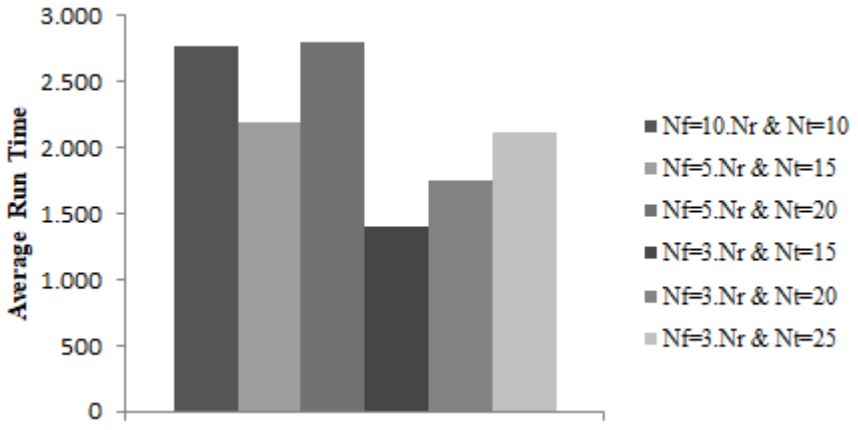



Figure 4.2. Average run time investigation of combinations $N_f$ & $N_t$

Table 4.9, Table 4.10, Table 4.11, Table 4.12, Table 4.13, and Table 4.14 contain the results of the trial experiments for the investigation of the best combination of parameters $N_r$, $N_f$, and $N_t$. In these trial experiments, it was aimed to increase the value of the Nt parameter while decreasing the value of the Nr parameter and the value of the Nf parameter in order to allow more runners to try more solutions. The parameter combination $(N_r, N_f, N_t) = (3.N_m, 4.N_r, 15)$ has better average final cost but its average run time is slightly more than the average run time of RRA as shown in Table 4.9. On the other hand, as Table 4.10 indicates the result of parameter set $(N_r, N_f, N_t) = (2.N_m, 4.N_r, 20)$ has also better average final cost and its average run times are barely better than the average run time of RRA. However, the result of parameter combination $(N_r, N_f, N_t) = (2.N_m, 3.N_r, 25)$ has better average final cost and its average run times are much better than the average run time of RRA.

In Table 4.12, Table 4.13, and Table 4.14, the results of trial experiments which have parameter sets $(N_r, N_f, N_t) = (N_m, 5.N_r, 40)$, $(N_r, N_f, N_t) = (N_m, 4.N_r, 45)$ and $(N_r, N_f, N_t) = (N_m, 3.N_r, 50)$ are shown respectively. In these three trial experiments, $N_r$ was chosen to be $N_m$ and the value of $N_t$ was increased while the value of $N_f$ was decreased. The average final cost and average run time of these three parameter sets are better than the average final cost and average run time of RRA. It is seen that there is a trade off between the final cost and the run time as a result of increasing Nt and decreasing Nf. The result of $(N_r, N_f, N_t) = (N_m, 5.N_r, 40)$ has much better average final cost and the result of $(N_r, N_f, N_t) = (N_m, 3.N_r, 50)$ has much better average run time among these three trial experiments. Besides, the result of $(N_r, N_f, N_t) = (N_m, 3.N_r, 45)$ has average final cost and average run time between the results of $(N_r, N_f, N_t) = (N_m, 5.N_r, 40)$ and $(N_r, N_f, N_t) = (N_m, 3.N_r, 50)$.

After all these trial experiments to determine the best values of $N_r$, $N_f$ and $Nt$, the best empirical parameter set is determined as $(N_r, N_f, N_t) = (3.N_m, 3.N_r, 20)$. Furthermore, the values of parameters $N_r$ and $N_f$ are adaptive according to $N_m$ in MRRA, while these parameters are fixed numbers in RRA. The parameter set $(N_r, N_f, N_t) = (3.N_m, 3.N_r, 20)$ was utilized for further experiments which includes experiments of other circuits in MCNC benchmark suite.

Table 4.9. Trial experiments for $N_r = 3.N_m$ & $N_f = 4.N_r$ & $N_t = 15$

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.319 | 1.939 | 2.558 |
| Best Cost | 5.160 | 1.238 | 1.777 | 2.201 |
| Best Time | 3.582 | 1.381 | 1.414 | 1.954 |
| Worst Cost | 3.531 | 1.382 | 2.201 | 3.044 |
| Median | 3.969 | 1.318 | 1.893 | 2.495 |

Table 4.10. Trial experiments for $N_r = 2.N_m$ & $N_f = 4.N_r$ & $N_t = 20$

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.318 | 1.734 | 2.287 |
| Best Cost | 4.611 | 1.253 | 1.582 | 1.983 |
| Best Time | 3.511 | 1.330 | 1.288 | 1.713 |
| Worst Cost | 3.891 | 1.383 | 1.787 | 2.473 |
| Median | 3.969 | 1.315 | 1.692 | 2.226 |

Table 4.11. Trial experiments for $N_r = 2.N_m$ & $N_f = 3.N_r$ & $N_t = 25$

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.312 | 1.515 | 1.989 |
| Best Cost | 2.970 | 1.239 | 1.576 | 1.953 |
| Best Time | 4.586 | 1.352 | 1.226 | 1.658 |
| Worst Cost | 3.361 | 1.391 | 1.329 | 1.849 |
| Median | 3.969 | 1.312 | 1.501 | 1.970 |

Table 4.12. Trial experiments for $N_r = 1.N_m$ & $N_f = 5.N_r$ & $N_t = 40$

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.309 | 1.793 | 2.349 |
| Best Cost | 3.846 | 1.248 | 1.821 | 2.274 |
| Best Time | 4.144 | 1.334 | 1.472 | 1.964 |
| Worst Cost | 5.092 | 1.374 | 1.697 | 2.333 |
| Median | 3.969 | 1.306 | 1.771 | 2.313 |

Table 4.13. Trial experiments for $N_r = 1.N_m$ & $N_f = 4.N_r$ & $N_t = 45$

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.315 | 1.691 | 2.225 |
| Best Cost | 3.225 | 1.246 | 1.716 | 2.138 |
| Best Time | 3.582 | 1.378 | 1.384 | 1.907 |
| Worst Cost | 3.361 | 1.389 | 1.569 | 2.180 |
| Median | 3.969 | 1.316 | 1.689 | 2.223 |

Table 4.14. Trial experiments for $N_r = 1.N_m$ & $N_f = 3.N_r$ & $N_t = 50$

| **Result** | **Initial Cost** | **Final Cost** | **Run Time** | **Cost x Time** |
|---|---|---|---|---|
| Average | 3.995 | 1.318 | 1.503 | 1.982 |
| Best Cost | 2.769 | 1.250 | 1.460 | 1.826 |
| Best Time | 4.372 | 1.308 | 1.246 | 1.630 |
| Worst Cost | 3.965 | 1.399 | 1.496 | 2.094 |
| Median | 3.969 | 1.316 | 1.481 | 1.949 |

Figure 4.3 represents the comparison in the average cost of parameter combinations in Table 4.9, Table 4.10, Table 4.11, Table 4.12, Table 4.13, and Table 4.14. As Figure 4.3 indicates the parameter combination in Table 4.12 has the best average cost and the parameter combination in Table 4.9 has the worst average cost.

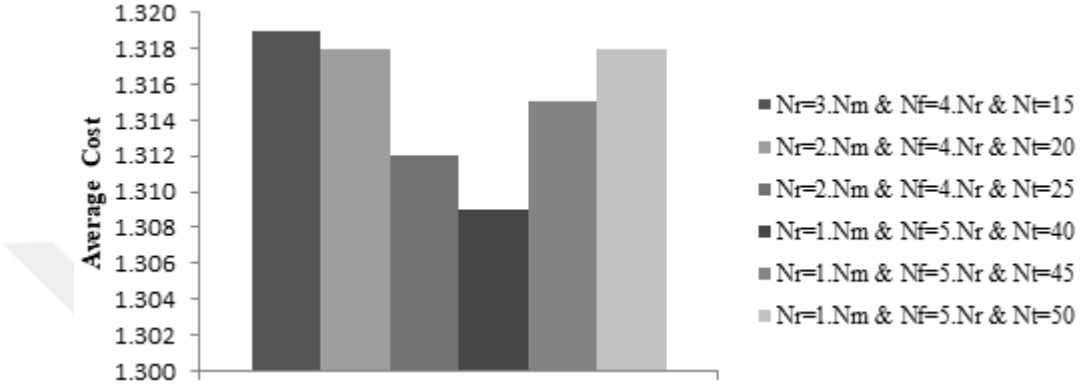



Figure 4.3. Average cost investigation of combinations $N_r$ & $N_f$ & $N_t$

Figure 4.4 represents the comparison in the average run time of parameter combinations in Table 4.9, Table 4.10, Table 4.11, Table 4.12, Table 4.13, and Table 4.14. As Figure 4.4 indicates the parameter combination in Table 4.14 has the best average run time and the parameter combination in Table 4.9 has the worst average run time.

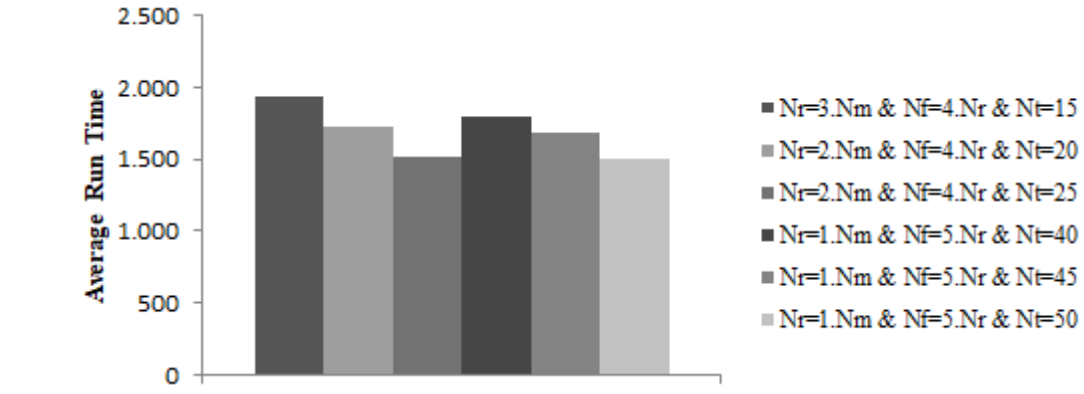



Figure 4.4. Average run time investigation of combinations $N_r$ & $N_f$ & $N_t$

## 4.2. Comparison

In this section, area optimization results of SA, GA, RRA, and MRRA are compared. For a fair comparison, all algorithms were implemented utilizing SP representation scheme in Java environment on 2.40 GHz PC with 8.00 GB memory. These algorithms were applied to all circuits found on the MCNC benchmark suite.

The values of the parameters used in the algorithms were chosen taking into account the values of the algorithms that the run times were close. For the implementation of RRA, $N_r$ and $N_f$ were selected as 100 and 1000 respectively, while $N_r = 3.N_m$ and $N_f = 3.N_r$ were chosen for the implementation of MRRA. On the other hand, different values were used for $N_t$ to provide close run times. The parameters of SA were set as follows: The initial temperature was determined by considering average difference cost of moving methods. The number of trials per each temperature and cooling rate were selected as 500 and 0.99 respectively. For the implementation of GA, the population size and the number of generation were set to 100 and 1000 respectively. In addition, crossover rate and mutation rate were chosen as 0.8 and 0.3 respectively.

The comparisons shown in Table 4.15, Table 4.16, Table 4.17, Figure 4.5, and Figure 4.6 are based on 10 trials for each algorithm. In Table 4.15, comparisons of average cost and average run time are shown respectively. RRA and MRRA have better results than SA and GA in both categories. Besides, MRRA has the best results for all benchmarks among these four algorithms. Table 4.16 illustrates the comparisons of best cost and worst cost. RRA and MRRA have also better results than SA and GA in these comparisons. On the other hand, RRA and MRRA have the same results for *apte* and the result of RRA is better than the result of MRRA for *ami*49. Nevertheless, MRRA has best results for the rest of the comparisons. As shown in Table 4.17, the comparison between MRRA and RRA indicates the improvement of both average cost and run time. The improvement of MRRA is from 0.3% to 1.7% for the average cost. In addition, MRRA has considerable improvement from 9.4% to 24.9% for the run time. The average improvement in run time is approximately 17.5%.

Table 4.15. Comparison in the average cost and average run time

| MCNC | Average Cost ($mm^2$) | | | | Average Run Time ($s$) | | | |
|---|---|---|---|---|---|---|---|---|
| | SA | GA | RRA | MRRA | SA | GA | RRA | MRRA |
| apte | 47,687 | 48.579 | 47.570 | 47.481 | 1.871 | 1.936 | 1.460 | 1.322 |
| xerox | 20,930 | 21.120 | 20.607 | 20.307 | 1.853 | 1.695 | 0.841 | 0.631 |
| hp | 9,971 | 9.822 | 9.528 | 9.361 | 2.779 | 1.662 | 1.102 | 0.928 |
| ami33 | 1,438 | 1.327 | 1.275 | 1.259 | 43.036 | 19.854 | 13.103 | 10.294 |
| ami49 | 46,256 | 41.907 | 39.714 | 39.702 | 85.255 | 36.799 | 27.404 | 22.587 |

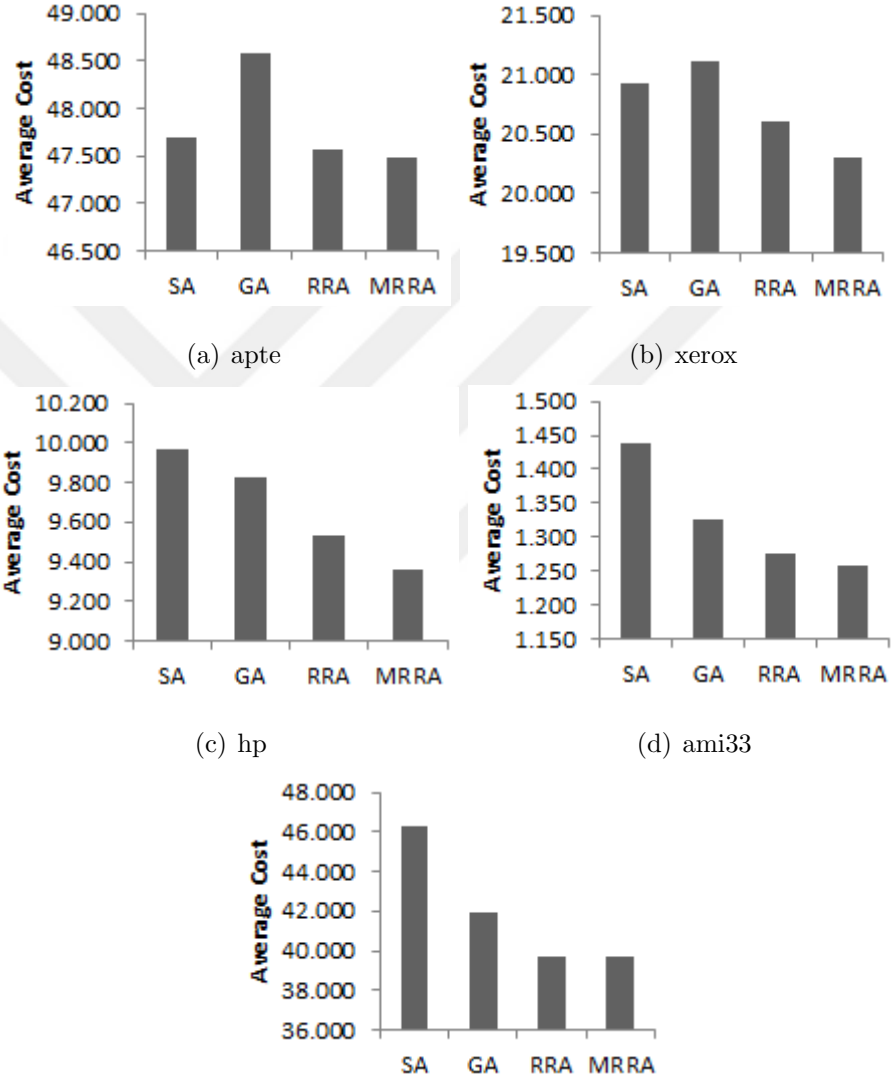
Table 4.16. Comparison in the best cost and worst cost

| MCNC | Best Cost ($mm^2$) | | | | Worst Cost ($mm^2$) | | | |
|---|---|---|---|---|---|---|---|---|
| | SA | GA | RRA | MRRA | SA | GA | RRA | MRRA |
| apte | 47,078 | 48.059 | 47.078 | 47.078 | 48.164 | 49.280 | 48.059 | 48.059 |
| xerox | 20,324 | 20.831 | 20.314 | 20.178 | 22.181 | 22.044 | 21.166 | 20.401 |
| hp | 9,560 | 9.631 | 9.208 | 9.144 | 10.434 | 10.224 | 9.879 | 9.613 |
| ami33 | 1,346 | 1.294 | 1.226 | 1.214 | 1.516 | 1.355 | 1.319 | 1.298 |
| ami49 | 43,909 | 40.322 | 39.112 | 39.030 | 50.207 | 43.268 | 40.532 | 41.000 |

Table 4.17. Improvement in the average cost and run time

| MCNC | Cost | | Run Time | | Improvement (%) | |
|---|---|---|---|---|---|---|
| | RRA | MRRA | RRA | MRRA | Cost | Run Time |
| apte | 47.570 | 47.481 | 1.460 | 1.322 | 0.187 | 9.452 |
| xerox | 20.607 | 20.307 | 0.841 | 0.631 | 1.455 | 24.970 |
| hp | 9.528 | 9.361 | 1.102 | 0.928 | 1.752 | 15.789 |
| ami33 | 1.275 | 1.259 | 13.103 | 10.294 | 1.254 | 21.437 |
| ami49 | 39.714 | 39.702 | 27.404 | 22.587 | 0.030 | 16.103 |

As Figure 4.5 indicates RRA and MRRA have better average cost than SA and GA, for the entire MCNC suite. In addition, MRRA has the best average cost among these algorithms for all circuits in MCNC benchmark.



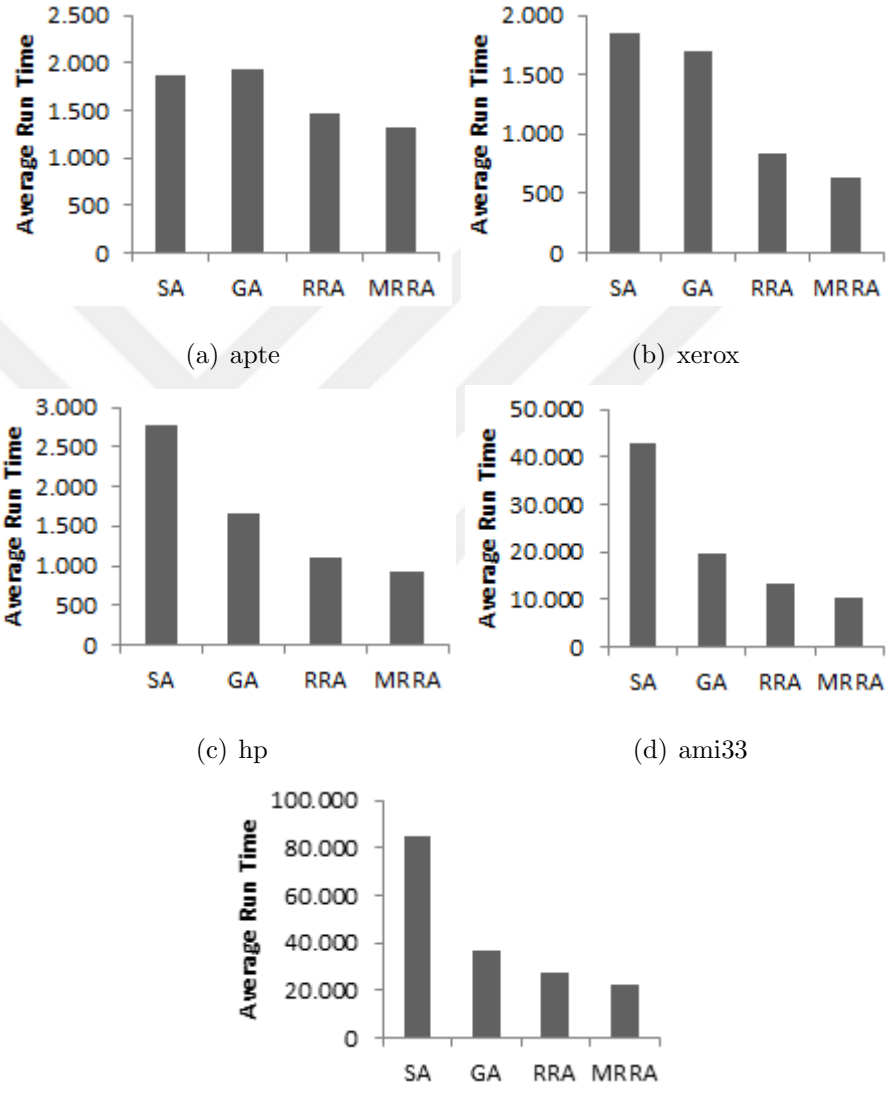

(a) apte (b) xerox

(c) hp (d) ami33

(e) ami49

Figure 4.5. Comparison in the average cost

As Figure 4.6 indicates RRA and MRRA have better average run time than SA and GA, for the entire MCNC suite. In addition, MRRA has the best average run time among these algorithms for all circuits in MCNC benchmark.




(a) apte (b) xerox

(c) hp (d) ami33

(e) ami49

Figure 4.6. Comparison in the average run time

# 5. CONCLUSION AND FUTURE WORK

In this thesis, a heuristic approach named Modified Relay Race Approach, is proposed to solve the floorplanning problem. MRRA was designed to improve the speed and quality of RRA by overcoming the shortcomings of RRA. In MRRA, dual path search is designed to increase the probability of exploring a better local optimal solution as next solution. In both search paths, rough run and focusing run are implemented and the only difference between these two paths is the percentage of randomly generated part of the solution $R_e$ in the relay operation. The dual path search has its own termination condition in order not to increase the run time. Moreover, parameters $N_r$ and $N_f$ are determined according to the detailed analysis, which also considers the number of modules in the circuit to improve the efficiency of algorithm. The efficiency of MRRA is approved by applying it to floorplanning problem in physical design optimization. Based on the comparisons of the experimental results utilizing MCNC benchmark suite, MRRA is better than SA, GA and RRA in the average cost and average run time of area optimization. MRRA reduced average run time by an average of 17.5% according to the RRA. With regard to comparison results, the proposed MRRA has potential to improve more NP-hard problems.

For the future work, changing the parameters during search and initialization from more than two path could be implemented in MRRA to enhance the search efficiency. As shown in the comparisons in Chapter 4, the improvement of MRRA varies according to MCNC benchmark. The difference in the number of modules of the circuits may be the cause of this situation. Although the parameter values used in the MRRA were determined as a result of a detailed analysis, they may need to be changed according to the region where they are located in search space. Searching by more than one path, as evidenced by MRRA, increases the efficiency. However, the most suitable number of initial path and their termination condition can be determined to increase the improvement of efficiency. In addition, these multiple paths can be operated on different cores to decrease the computation time.

# REFERENCES


1. Singh, R. B., A. S. Baghel and A. Agarwal, "A Review on VLSI Floorplanning Optimization Using Metaheuristic Algorithms", *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 4198–4202, March 2016.

2. Banerjee, S., A. Ratna and S. Roy, "Satisfiability Modulo Theory Based Methodology for Floorplanning in VLSI Circuits", *2016 Sixth International Symposium on Embedded Computing and System Design (ISED)*, pp. 91–95, December 2016.

3. Kirkpatrick, S., C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, pp. 671–680, 1983.

4. Sheng, Y., A. Takahashi and S. Ueno, "Relay-Race Algorithm: A Novel Heuristic Approach to VLSI/PCB Placement", *2011 IEEE Computer Society Annual Symposium on VLSI*, pp. 96–101, July 2011.

5. Wong, D. F. and C. L. Liu, "A New Algorithm for Floorplan Design", *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pp. 101–107, IEEE Press, 1986.

6. Rebaudengo, M. and M. S. Reorda, "GALLO: A Genetic Algorithm for Floorplan Area Optimization", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 8, pp. 943–951, August 1996.

7. Zhou, Y., Y. Yan and W. Yan, "A Method to Speed Up VLSI Hierarchical Physical Design in Floorplanning", *2017 IEEE 12th International Conference on ASIC (ASICON)*, pp. 347–350, October 2017.

8. Sowmya, B. and M. Sunil, "Minimization of Floorplanning Area and Wire Length Interconnection Using Particle Swarm Optimization", *International Journal of*

*Emerging Technology and Advanced Engineering*, Vol. 3, No. 8, 2013.

9. Arumugam, S., "Certain Optimization Techniques for Floorplanning in VLSI Physical Design", *Chennai*, 2009.

10. Ray, B. N. B., A. R. Tripathy, P. Samal, M. Das and P. Mallik, "Half-Perimeter Wirelength Model for VLSI Analytical Placement", *2014 International Conference on Information Technology*, pp. 287–292, December 2014.

11. Laskar, N. M., R. Sen, P. K. Paul and K. L. Baishnab, "A Survey on VLSI Floorplanning: Its Representation and Modern Approaches of Optimization", *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1–9, March 2015.

12. Nakatake, S., K. Fujiyoshi, H. Murata and Y. Kajitani, "Module Placement on BSG-Structure and IC Layout Applications", *Proceedings of International Conference on Computer Aided Design*, pp. 484–491, November 1996.

13. Lin, J.-M. and Y.-W. Chang, "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans", *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pp. 764–769, 2001.

14. Murata, H., K. Fujiyoshi and M. Kaneko, "VLSI/PCB Placement with Obstacles Based on Sequence Pair", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, No. 1, pp. 60–68, January 1998.

15. Xiaogang, W., "VLSI Floorplanning Method Based on Genetic Algorithms", *Microprocessors*, Vol. 1, p. 1, 2002.

16. Chen, G., W. Guo, H. Cheng, X. Fen and X. Fang, "VLSI Floorplanning Based on Particle Swarm Optimization", *2008 3rd International Conference on Intelligent System and Knowledge Engineering*, Vol. 1, pp. 1020–1025, November 2008.

17. Luo, R. and P. Sun, "A Novel Ant Colony Optimization Based Temperature-Aware Floorplanning Algorithm", *Third International Conference on Natural Computation (ICNC 2007)*, Vol. 4, pp. 751–755, August 2007.

18. Moni, D. J., S. Arumugam and G. N. Rani, "Vlsi Floorplanning Relying on Differential Evolution Algorithm", *ICGST International Journal on Artificial Intelligence and Machine Learning*, Vol. 7, No. 1, pp. 62–67, 2007.

19. Otten, R., "Efficient Floorplan Optimization", *Proc. ICCD, 1983*, 1983.

20. Hajek, B., "Cooling Schedules for Optimal Annealing", *Mathematics of Operations Research*, Vol. 13, No. 2, pp. 311–329, 1988.

21. Mani, N. and B. Srinivasan, "Using Genetic Algorithm for Slicing Floorplan Area Optimization in Circuit Design", *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, Vol. 3, pp. 2888–2892 vol.3, October 1997.

22. Nakaya, S., T. Koide and S. Wakabayashi, "An Adaptive Genetic Algorithm for VLSI Floorplanning Based on Sequence-Pair", *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings*, Vol. 3, pp. 65–68, 2000.

23. Lin, C. T., D. S. Chen and Y. W. Wang, "An Efficient Genetic Algorithm for Slicing Floorplan Area Optimization", *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, Vol. 2, pp. II–II, IEEE, 2002.

24. Gwee, B.-H. and M.-H. Lim, "A GA with Heuristic-Based Decoder for IC Floorplanning", *INTEGRATION, the VLSI journal*, Vol. 28, No. 2, pp. 157–172, 1999.

25. Drakidis, A., R. J. Mack, R. E. Massara, A. Drakidis, R. J. Mack and R. E. Massara, "Packing-Based VLSI Module Placement Using Genetic Algorithm with Sequence-Pair Representation", *IEE Proceedings - Circuits, Devices and Systems*,

Vol. 153, No. 6, pp. 545–551, December 2006.