論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	High-Performance Heuristics with Applications to 2D/3D IC Physical Design Optimization
著者(和文)	盛益強
Author(English)	Yiqiang Sheng
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第9602号, 授与年月日:2014年7月31日, 学位の種別:課程博士, 審査員:高橋　篤司,國枝　博昭,上野　修一,一色　剛,原　祐子,小平　行秀
Citation(English)	Degree:, Conferring organization: Tokyo Institute of Technology, Report number:甲第9602号, Conferred date:2014/7/31, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctoral Dissertation

High-Performance Heuristics with Applications to 2D/3D IC Physical Design Optimization

Tokyo Institute of Technology
Graduate School of Science and Engineering
Department of Communications and Integrated Systems
Ueno-Takahashi Lab

March, 2014

Student ID : 10D51393
Name : Yiqiang Sheng
Supervisor : Atsushi Takahashi

博士学位論文

高性能最適化手法と 2D/3D 集積回路物理設計最適化への応用

東京工業大学
理工学研究科
集積システム（通信情報工学）専攻
上野・高橋研究室
平成 26 年 3 月

学籍番号： 10D51393
氏名： 盛 益強
指導教官： 高橋 篤司

Abstract

High-performance heuristic is a class of search method which improves the efficiency of NP-hard problem solving in many different technical fields such as 2D/3D IC physical design optimization. With the fast advancement of design optimization, it is increasingly hard to satisfy the high-performance requirements of computation due to short product life cycle, large design scale and high complexity. In this research, various techniques including moving method, adaptive selection and similar optimum search are discussed to improve the performance of existing search methods. Based on the combination of the techniques, two new heuristic algorithms are proposed as follows. Firstly, a new variation of simulated annealing named adaptive simulated annealing with crossover (ASA_X) is proposed by introducing a guide with adaptive probabilities to select diverse moving methods including a special crossover. Secondly, a novel heuristic named relay-race algorithm (RRA) is proposed to explore similar local optimal solutions more efficiently by introducing rough moving methods and relay with sophisticated strategies. RRA includes three stages: rough search, focusing search and relay. Rough search is designed to get over small hills on the solution space and to approach a local optimal solution as fast as possible. In rough search, rough moving methods that modify a current solution relatively large are used. The focusing search is designed to reach the local optimal solution as close as possible. In focusing search, focusing moving methods that modify a current solution little are used. Relay is one-time modification which is controlled by a global parameter so that the solution generated is far enough from the current local optimum to escape from it without loss of search continuity. As typical applications, the proposed heuristics are used to solve packing and placement problems in 2D/3D IC physical design. According to the experimental data, the performance is considerably improved. The overall Pareto improvement of two conflicting objectives is obtained. With regard to the impact, the proposed heuristics have potential to improve existing search methods for more NP-hard problems.

概要

多項式時間で最適解を得ることが困難であると考えられているＮＰ困難と呼ばれる様々な最適化問題に対して、様々な種類のアルゴリズムが提案されている。集積回路物理設計などの分野では短時間で準最適解を得ることが要求されるが、問題規模の増大などにより、短時間に準最適解を得ることが困難となってきており、アルゴリズムのさらなる性能向上が求められている。本研究では、探索的アルゴリズムの性能を向上させる技法を提案し、それらを適切に組み合わせることで新たな高性能な探索的アルゴリズムASA-XおよびRRAを提案した。ASA-X 法は、SA の拡張アルゴリズムで、SA の基本的な修正操作に加え最良解との交叉を取り入れ、適応的選択技法を用いることで準最適解を効率よく得る。交叉を導入することで初期段階での大域的探索の効率を上げるとともに適応的選択技法を用いることで、最終段階での探索効率の悪化を防ぐ。RRA法は、概略探索、詳細探索、リレーを繰り返すことで準最適解を効率よく得る。RRA の概略探索および詳細探索は、改善する隣接解のみを選択する逐次改善法を構成し、概略探索では解を大きく変更する修正操作を用いて良解の近傍まで効率よく到達し、詳細探索では解をちいさく変更する詳細修正により良解まで効率よく到達する。リレーでは、ランダム解を適切な割合で現在解に交叉させることで、局所最適解から抜け出し、近傍の他の良解を探索することを可能とする。多くの良解を効率よく探索することで、準最適解を効率よく得る。実験では、ASA-X および RRA を 2 次元および 3 次元の集積回路物理設計問題に適用し、既存アルゴリズムと比較することで、短時間に良解を得られることを確認した。提案技法および提案アルゴリズムは、様々な分野の大規模な最適化問題に対して効果が得られることが期待される。

Contents

1. Introduction

Search methods such as [1-19] are widely used to get near-optimal solutions for NP-hard problems. As shown in Figure 1, existing search methods include exhaustive search (ES), local search (LS), random search (RS), heuristics, etc. Wherein, heuristic is an empirically effective search method constructed by using known successful experiences of problem solving. Random search, also known as trial-and-error method, could be regarded as a simplified heuristic without consideration of successful experiences. Most heuristics are effective in practice but there is no theoretical proof on their efficiencies. High-performance heuristic is a class of heuristic which improves the efficiency of problem solving in many different technical fields such as 2D/3D IC physical design optimization where existing search methods have failed to be efficient or effective in practice.

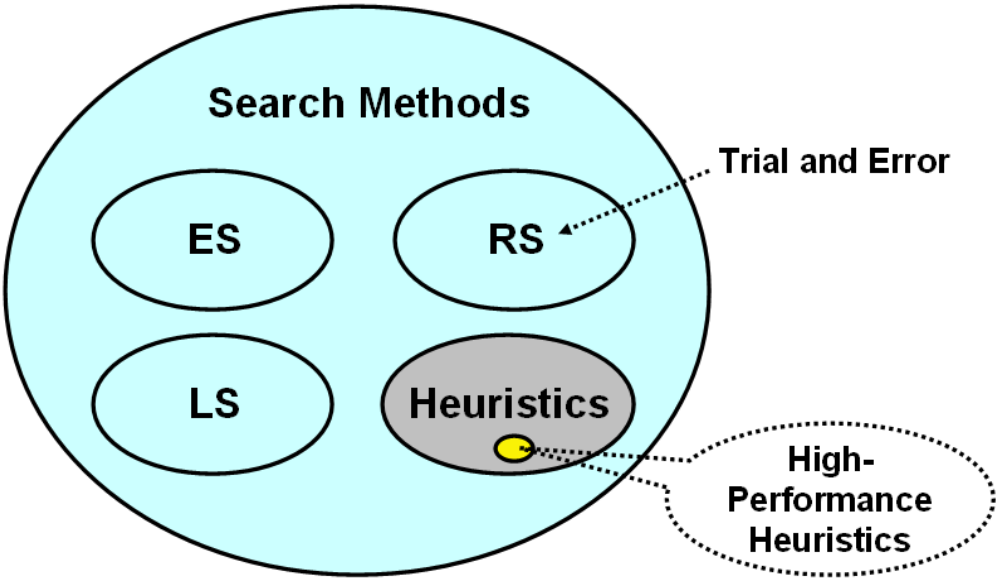

Figure 1: Search methods include exhaustive search (ES), local search (LS), random search (RS), heuristics, etc.

This chapter consists of four sections. The first section briefly introduces the hardness of problems which can be defined by the complexity of algorithms using an

ideal computer model. In this section, NP-Hard problem as well as its importance is introduced. The second section is on heuristic methods for problem solving. As a typcial modern heuristic algorithm, simulated annealing is analyzed in details. The last two sections present high-performance requirements of heuristic algrithms as well as main contributions of this research.

1.1 Hardness of Problems

The purpose of this research is to solve hard problems more efficiently. First and foremost, it is not easy to define the exact hardness of a given problem. In practice, it is normally subjective to judge the hardness of a general problem because it depends on the people who solve it, the past experience of the solvers and how much time to solve it. If the human factor has to be considered, it is only possible to have some statistical data as the statistical criterion for some special population. In order to have non-subjective criterion, one of the foremost things is to exclude the human factor. An alternative way is trying to solve the given problem by a common computer instead of diverse people.

Why a problem is hard even by using computer? From empirical point of view, there are at least two reasons as follows. The first reason is the solution space is so large that it is not possible to search all feasible solutions using exhaustive search (ES) within a given computational time even using the fastest-so-far computer in the world. The so-called solution space generally means feasible solution space in which all constraints of problem are satisfied, though it is not limited to feasible solution space in some special search strategies. The solution space could be infinite or so large that we do not know how large it is. The second reason is the state-cost distribution is so complex that it is not possible to get the global optimum by continuous improvement using local search (LS) using known models or rules. That is to say, any problem which can be solved by exhaustive search or local search within a given runtime is not practically hard.

Besides, there is another way to judge a given problem is not hard in practice. It

is tested by trial-and-error method which is widely used in science and technology. In the trial-and-error method, every problem is regarded as a black box, i.e. a system that is totally unknown inside. It is always operable to have a random trial and to get a response. According to the expectation of response, we can always judge that the response is an error or not, i.e. YES/NO answer of a decision-version problem, by comparison between response and expectation. Furthermore, we can get to know how close between response and expectation, i.e. the answer of an optimization-version problem. Actually, trial-and-error method is a general method which can be used to get an answer of any problem, though it is inefficient to solve most of hard problems. In short, any problem which can be solved by the trial-and-error method, i.e. random search (RS), within a given runtime is not practically hard.

According to theory of combinatorial optimization [20], the hardness of a decision-version problem can be measured by the complexity of best-so-far algorithm to solve the problem using an ideal computer model such. The best-so-far algorithm means the most efficient one among known efficient algorithms. However, it is still not easy to judge the hardness of a general problem since there are too many different versions of problem. As one of the simplest versions of problem, the decision-version problem whose answer is only limited to “YES” or “NO” is a good start to classify the hardness of problems.

The complexities of algorithms to solve decision problems are classified as P class and NP class, though we do not know that P equals to NP or not so far (by 2013). Based on the complexities of the most efficient algorithms, the hardness of decision problems can be classified as P class and NP class under the assumption of $P \neq NP$, as shown in Figure 2. It is theoretically possible for a given problem in P class to answer “YES” or “NO” in polynomial time by using existing algorithms, while it is theoretically possible for a given problem in NP class to check whether a given solution is “YES” or “NO” in polynomial time by using existing algorithms. The decision problem in NPC class is the hardest problem in NP class. If there is an algorithm to solve any problem in NPC class in polynomial time, it would be theoretically possible for any given problem in NP class to find the final solution in

polynomial time by using the algorithm, i.e. the proof of NP=P. Otherwise, there are no such efficient algorithms that solve a problem in NPC class in polynomial time, it would be theoretically impossible for any given problem in NP class to find the final solution in polynomial time by using the algorithm, i.e. the proof of NP≠P. That is the famous P/NP puzzle.

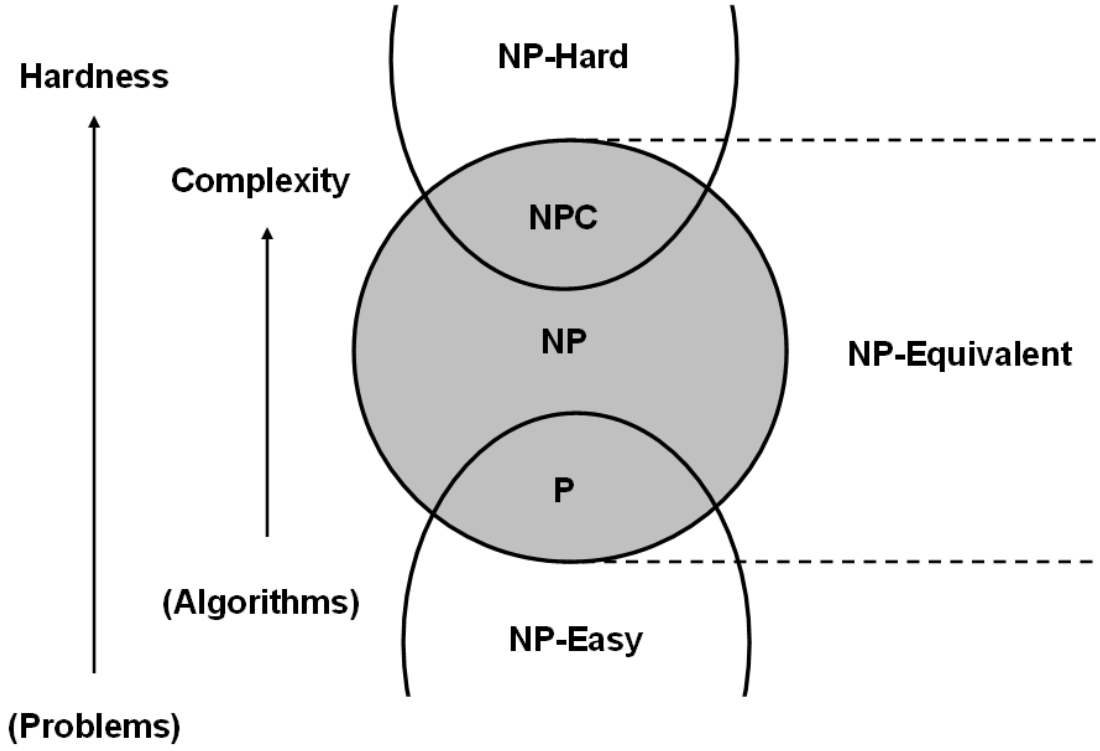

Figure 2: From the complexity of algorithms to the hardness of problems

(Assumption: P ≠ NP)

Furthermore, the decision version of problem is extended to other versions such as the search version of problem and the optimization version of problem. The so-called search problem is to find a "YES" answer with at least one of feasible solutions when the answer of the decision problem is "YES". The so-called optimization problem is to find at least one "optimal" solution which is the best of all feasible solutions, when the answer of the decision problem is "YES". For any versions, if a problem is as hard as the decision problem in NP class, it belongs to the class of NP-Equivalent. If a problem is at most as hard as the decision problem in P

class, the problem belongs to the class of NP-Easy. If a problem is at least as hard as the decision problem in NPC class, the problem belongs to the class of NP-Hard, as shown in Figure 2.

In fact, any optimization problem is in the class of NP-Hard, if the corresponding decision version of the problem is in the class of NPC, because the optimization problem is at least as hard as the NPC decision problem. Moreover, if there is an algorithm to solve any problem in the NP-Hard class in polynomial time, it would be theoretically possible to solve any given problem in the class of NP and NP-Equivalent in polynomial time by using the algorithm. The applications of this research are focusing on solving NP-Hard optimization problems more effectively.

In general, it is safe to say that all practical problems are come from the gap between expectation and reality. In the modern society, the expectation of people is becoming increasingly higher, but the reality in the world is becoming more complex due to the advancement of technologies and the frequent interaction among people. Besides, the nonlinearity increases the hardness of practical problems.In fact, there are not exact algorithms so far to solve a general nonlinear optimization problem within polynormimal time. Let us take the interaction as an example. In order to realize our expectation, the best strategy of interaction depends on our objectives and other people's strategies. More and more factors have to be considered to solve a practical problem due to the conflicting objectives among more people. In fact, almost all practical problems are becoming harder to be solved due to the higher complexity of reality as well as the higher expectation of human being.

1.2 Heuristics for Problem Solving

As one of the most pragmatic search methods, heuristic algorithm is constructed to find near-optimal solutions according to the experiences which have been proved to be effective or efficient in practice. Based on the past experiences, a series of heuristic rules are introduced to improve the efficiency of search process. The heuristic rules could be predetermined fixed rules, dynamical rules or priority rules. In general, a

search method is not only dedicated to solve the search problem but also used to solve other versions of problem. A decision problem is solved as "YES" answer if any feasible solution has been checked by search method. An optimization problem is also solved if all feasible solutions have been checked to find a global optimum by search method. In fact, many important discoveries in science have a direct relationship with heuristic methods.

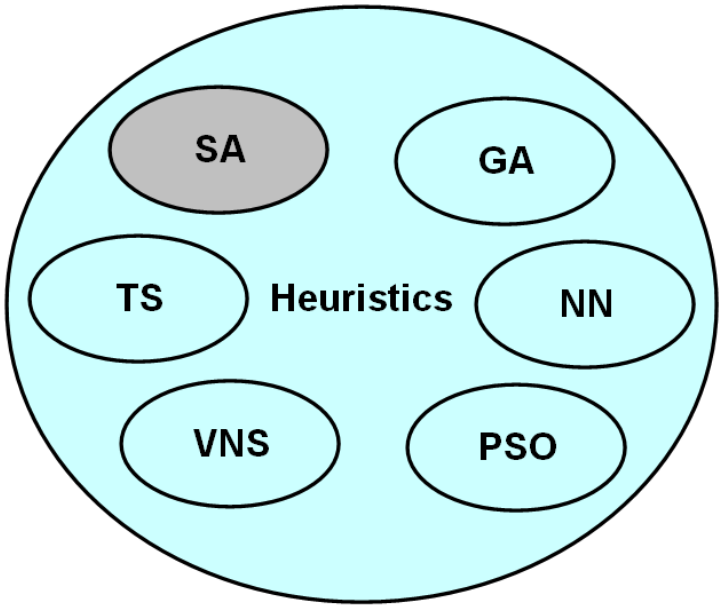

Figure 3: Heuristics include simulated annealing (SA), tabu search (TS), variable neighborhood search (VNS), genetic algorithm (GA), neural network (NN), particle swarm optimization (PSO), etc.

Trial-and-error method, i.e. random search or learning from mistakes, is regarded as a simplified heuristic algorithm without heuristic rules and past experiences. Nowadays, it is still one of essential features of heuristics to try some possibilities by trial-and-error method, though it is too inefficient to try all possibilities. If some self-discovery rules are added to guide the search process toward a more effective search direction, it is the prototype of modern heuristics, metaheuristics or super heuristics.

Various heuristic algorithms, such as simulated annealing (SA) [11-13], tabu

search (TS) and variable neighborhood search (VNS), genetic algorithm (GA) [14], neural network (NN) [15] and particle swarm optimization (PSO) [16] as shown in Figure 3, have been proposed to solve practical problems. Wherein, most heuristics are based on the analogy of artificial intelligent (AI) systems including human brain, biological system, social system, and so on. For example, the neural network is a simplified model of human brain. The genetic algorithm is a simulated process of genetic evolution. The particle swarm optimization is a simulation system of swarm intelligence and group behavior. It is not only limited to solve computing problems by the biotechnological analogy but also used to do the inverse researches of some biological phenomena by the computing simulation.

From the optimization point of view, all search methods for a given problem are trying to find the global optimum in solution space as effectively as possible based on the known facts and past experiences within given time and resources. In the solution space, the adjacency between solutions is defined by moving methods that modify one solution to another solution. A local optimal solution is a solution whose cost is not worse than any adjacent solutions. Whether a solution is a local optimal solution or not depends on the adjacency defined by moving methods. That is, a local optimal solution defined by a set of moving methods is not necessarily a local optimal solution defined by another set of moving methods. In typical search algorithms, an adjacent solution is randomly generated by a moving method. If the adjacent solution satisfies some condition, then the current solution is replaced with it. Otherwise, it is rejected, and the current solution is kept.

An iterative improvement method rejects an adjacent solution if it is worse than the current solution. It stops when it regards the current solution as a local optimal solution. However, the final solution obtained is often far worse than a global optimal solution, which is the best of all local optimal solutions. In order to get a better solution efficiently, various ideas such as iterative improvement with random generation and simulated annealing have been introduced. When iterative improvement is repeated from randomly generated initial or restarted solutions, the probability of finding a better solution becomes larger. However, an obtained local

optimal solution is independent of others. A local optimal solution which is similar to an obtained local optimal solution would not be explored. It is expected that a better local optimal solution exists which is similar to an obtained local optimal solution. The efficiency is not good enough since the search history is not utilized.

As one of the most pupolar heuristic algorithms, simulated annealing (SA) accepts a non-improved adjacent solution with some probability according to temperature scheduling to escape from a local optimal solution. In order to obtain a better solution by simulated annealing, a solution space that satisfies the following properties is usually desired to be constructed. 1) The solution space is connected. This guarantees the existence of a search path from any initial solution to an optimal solution by a finite number of iterations. 2) The diameter of the solution space is small. This gives a chance to reach an optimal solution by a small number of iterations in best cases. 3) The difference of costs between adjacent solutions is small. This improves the stability and convergence of a search algorithm.

In a solution space used in simulated annealing, if the number of adjacent solutions is large, then the connectivity and small diameter would be realized, and the number of local optimal solutions tends to be small. However, it is not easy to confirm whether a solution is local optimum. Also, the difference of costs of adjacent solutions tends to be large and the ratio of better solutions among adjacent solutions tends to be small. The probability that a generated adjacent solution is accepted becomes quite low. Therefore, in simulated annealing, a set of moving methods is designed so that the difference of costs of adjacent solutions is not large. In general, a moving method that may cause a drastic solution change is not used. This feature limits a search ability of simulated annealing since the number of required iterations including rejection from a local optimum to another local optimum is large. Even if a solution space is explored globally in earlier stage, similar local optimal solutions are not efficiently explored.

To improve the existing search methods further, various emerging techniques including moving method, adaptive selection and similar optimum search are discussed and proposed. Based on the techniques, two novel high-performance

heuristics, named adaptive simulated annealing with crossover (ASA_X) and relay-race algorithm (RRA), are proposed with applications to 2D/3D IC physical design optimization. As shown in Table 1, different techniques are used in different heuristics as follows. The technique of moving method is used in SA, GA, ASA_X and RRA, while the diverse moving methods are used to ASA_X and RRA. The technique of adaptive selection is used to ASA_X. The technique of similar optimum search is dedicated to RRA.

Table 1: Different techniques are used in the following heuristics: simulated annealing (SA), genetic algorithm (GA), adaptive simulated annealing with crossover (ASA_X) and relay-race algorithm (RRA).

Various Techniques	**SA**	**GA**	**ASA_X**	**RRA**
Moving Method	Yes	Yes	Yes	Yes
Diverse Moving Methods	No	No	Yes	Yes
Adaptive Selection	No	No	Yes	No
Similar Optimum Search	No	No	No	Yes

1.3 Requirement of High Performance

With the fast advancement of product design, it is increasingly hard to satisfy the high-performance requirements of product due to its short life cycle, large design scale and high complexity. A general product could be anything that is man-made. A complex product is defined as an integrated system with special functions to satisfy the requirements of customers. A product is regarded as an unnatural system, therefore the development of product comes down to system enginnering. The goal of system enginnering is to achieve overall optimal design for a complex and interdisciplinary development project. That is to say, the design optimization is the core of system engineering.

It is hard to achieve optimal, near-optimal or acceptable product designs without

high-performance computation. An essential reason is because the problem of product design optimization is NP-hard with conflicting objectives which includes functionality, quality, security, reliability, etc. If the designer trys to improve one of multiple objectives, it is probably to impact another objective. The designer has to face a situation of tradeoff between two or more conflicting objectives. For example, it is common that a faster electronical device tends to take larger power consumption without the consideration of technical breakthrough. It is a big challenge to improve speed and reduce power consumption simultaneously. An important concept about multi-objecive optimization is Pareto improvement which is to improve one of multiple objectives without degradation of other objectives. Multiple objectives make the problem much harder to be solved in practice.

Let us take very large scale integation (VLSI) design as a typical example. VLSI design is divided into system abstract, logic design and physical design. System abstract includes specification, architecture design and function design. Logic design is divided into different levels including system level, register-transfer level, cicuit level and gate level. Physical design includes partitioning, chip planning, floorplanning, placement, clock synthesis, routing, etc. The input of physical design is circuit diagrams, and the output is fabrication data as the terminal of design. From the system point of view, VLSI chip is a subsystem of printed circuit board of electronic products such as mobile phone. In fact, an efficient design of any complex system requires a necessary decomposition of subsystems. A complex electronic system generally consist of many different functional modules, such as central processing unit (CPU), memory storage module, power management module, etc. Furthermore, the functional modules may be divided into submodules or basic components such as transistors, diodes, resistors, capacitors, etc. All subsystems, modules, components and interconnect lines are supposed to be integrated into a semiconductor substrate as a chip, i.e. so-called system on chip (SoC).

For system design optimization, a simplified problem to be solved is how to arrange the location and the rotation of each component or each module, including each chip on printed ciruit board (PCB). That is so-called packing problem which is a

fixed-shape version of floorplanning. The input to floorplanning after circuit partitioning is a set of blocks, the area of each block, possible shapes of each block and the number of terminals for each block and the netlist. Furthermore, all mentioned system or subsystem is constituted by the interconnection between modules or components. That is so-called placement problem which is a packing problem with the consideration of interconnection. During the process of placement, the modules and the components are exactly positioned and rotated to meet specifications, and the interconnection is completed in the routing phase according to the netlist and the output of placement. In fact, the quality of placement would not be evident until the routing phase has been actually completed. A good routing depends on a good placement due to the fact that very small changes can be done to improve the routing and the overall performance once the location and the rotation of each module are fixed. A bad placement may lead to an unroutable design with in the provided space, so an estimation of the required routing space is necessary during placement. The objectives of placement normally include area minimization, routability, timing, power, etc. As a simplified but general model of system design, the placement occurs at different design levels including system level, board level and chip level.

It is possible to get optimal design by the manual work of product designers with their knowledge and experiences only if the scale of components or modules is small enough. However, with the scale becomes larger and larger nowadays, the situation of design optimization is completely changed. It is almost impossible to get optimal or even feasible design manually. The computer-aided or computer-automated design is going to be a necessity to get higher performance. For example, VLSI products are required with the progress of huge computational data processing in the technical field of multi-objective optimization, system engineering, fuzzy system design, dynamic system design optimization, and so on. Especially, with the progress of deep sub-micron and nano fabrication technology of integration circuits (ICs), many VLSI products have been worked in increasingly high speed, low power and high reliability. The requirements of high-performance computation are increased further due to this progress.

Therefore, it is unprecedentedly urgent and important for high-performance computation to solve hard optimization problems effectively. Without loss of generality, the packing problem is a typical hard problem with single objective, while the placement problem is a typical hard problem with multiple objectives. Both problems are important enough because there are many applications on different fields such as electronic design automation (EDA) for VLSI and PCB. Both problems are hard enough because they have been proved to be NP-hard. That is, there is no exact algorithm so far to get optimal solutions within polynomial time. Fortunately, the optimal solutions are not always necessary in practice. The key point is how to get near-optimal or acceptable solutions effectively within a given runtime by using high-performance search methods.

1.4 Main Contributions

The main contributions include several search techniques, a new variation of simulated annealing and a novel high-performance heuristic algorithm to solve practical problems more efficiently. Based on simulated anneaning and genetic algorithm, the techniques of moving methods and adaptive selection are discussed. Diverse moving methods are proposed to improve existing moving methods. An adaptive guide with a special crossover operator is proposed. As a new technique, similar optimum search is proposed at the first time to improve existing search methods by focusing on simlar local optimal solutions. Local optimum search is discussed as follows. State, move and cost are three elements of local search. State-cost distribution is an intuitive image to design efficient moving methods. Move-improve loop is a basic process to search local optimum by using moving methods. Multi-stage improvement and focus-and-relay loop are two parts of similar optimum search. Based on the similarity and diversity of local optimal solutions, multi-stage problem solver and heuristic thinking are discussed.

In this research, two new heuristic algorithms are proposed as follows. First of all, an improved heuristic algorithm, named adaptive simulated annealing with crossover

(ASA_X), is proposed to improve traditional simulated annealing by introducing special crossover and adaptive guide which is dedicated on the selection of moving methods. As a typical application of single objective optimization, ASA_X is used to solve 2D/3D packing problem and a big improvement is obtained. Furthermore, as a novel high-performance heuristic algorithm, relay-race algorithm (RRA) is proposed to explore similar local optimal solutions efficiently by introducing rough moving methods and relay operation with sophisticated strategy. Even if the solution space using basic moving methods is unconnected, it is still effective to approach the global optimal solutions. The proposed RRA consists of three stages: rough search, focusing search and relay. For the rough search, each perturbation is to move the state with some distance from the previous state. For the focusing search, each perturbation is limited in the local search. As a result of the focusing search, the state is dropped into the local optimum. Then, the relay is used to escape the local optimum and return to the rough search. In the step of relay, the next state is far enough from the local optimum in which the distance of relay is decided by the whole solution space without loss of search continuity. As typical applications, the proposed heuristics are used to solve packing problem and placement problem in 2D/3D IC physical design. According to the experimental data, the performance is considerably improved. The overall Pareto improvement of two conflicting objectives is obtained.

In essence, this research is about the art of problem solving with similarity. The focus of this research is on the method rather than the knowledge. Its purpose is to think about the problem of how to solve hard problem similarly and efficiently. In the real world, all problems as well as problem solving are similar. Problem exists forever if there is a gap between reality and expectation, and the problem solving is to eliminate or reduce the gap. On the side of reality, people always get what they deserved but not always what they wanted. On the side of expectation, a more efficient search method means more chance to approach what people wanted. In this research, intuitive heuristic ideas are summarized as how to think similarly. Multi-stage problem solver is about how to solve problems similarly and efficiently, while the heuristic thinking chains are about how to think about the problem of how to

solve problems similarly and effectively.

2. Preliminaries

This chapter mainly on optimization and engineering design [21-30] with applications to IC physical design [31-59]. It consists of three sections. The first section is a brief introduction of exisiting search methods. The second section introduces modern heuristic algorithms, including simulated annealing, genetic algorithm, tabu search, variable neighborhood search, neural network and particle swarm optimization. In the last section, the application to design optimization is discussed. Product, system and physical design optimizaiton is generally introduced. 2D/3D IC physical design optimizaiton is discussed in details. Especially, packing and placement optimizaiton problems are introduced as two typical applications of search methods.

2.1 Basic Search Methods

The basic description of an optimization problem consists of three elements: objectives, variables and constraints. The objective is the basic element which is evaluated by objective function, evaluation function or cost function to be minimized or maximized. In most optimization problems, the objective depends on the set of variables $\{x_i\} = \{x_1, x_2, \ldots, x_n\}$ which people can control and adjust to find optimal values of cost function. The change of variables is not entirely free but is subject to some constraints. The constraints are the restrictions arising from the nature of the problem. The variables and the constraints are not always necessary in theory, and the objective is the only necessary element of all optimization problems.

The basic description of a search method consists of three elements: state space $\{S_j\}$, a set of moving method $\{M_j\}$ and cost function $\{C_j\}$. The state space $\{S_j\}$ is a search space which includes all feasible solutions and sometimes infeasible solutions in the case of special search strategies. The feasible solution means a solution which is

satisfied with all constraints of the problem. The moving method M_j is a method which changes one solution to another solution in the state space $\{S_j\}$. It is not always necessary in some special search methods such as random search. The cost function $\{C_j\}$ is a function which evaluates the quality of a state or describes how close moving to the ideal state, i.e. the objective of an optimization problem. The cost function could be simply the difference between the current state and the ideal state. In fact, the state space $\{S_j\}$, i.e. solution space, is the only necessary element of a search method.

The objective and its cost function are two different concepts. The objective is an element of an optimizaiton problem, but the cost function $\{C_j\}$ is an element of a search method. Sometimes, the cost function is not always necessary. For example, an optimization problem is represented by (S_I, M_j, S_F), where S_I is the initial state, M_j is the moving method and S_F is the final state as the objective. A simple state space search without cost function can be used to solve such kind of problems directly. The basic process of simple state space method to solve a problem is as follows. It starts with some initial state S_I. A selected moving method M_j operates the initial state to get a current state S_j and repeatedly operates until reaching the final state S_F. In fact, the above state space can also be assigned to a graph. In the graph, a vertice represents a state, and an edge represents a move. That is, the problem is to search a path from the initial vertice S_I to the final vertice S_F which consists of many edges using moving method M_j.

In this section, we start with exhaustive search to solve the problem with small solution space which can be exhaustively searched within a given time. Then local search is introduced to solve the problem with simple state-cost distribution in which the number of local optimal solutions is small. And then random search is introduced to solve a general problem feasibly in theory but inefficiently in practice as a black box in which the inside information is totally unknown. Based on random search, heuristics are introduced to solve practical problems more efficiently by using past sucessful experience.

2.1.1 Exaustive Search and Simplex method

The basic method to get a global optimum is exaustive search. In exaustive search, all possible solutions in the state space $\{S_j\}$ are checked and evaluated one by one, and then output one or more than one of the best solutions. For designing an exaustive search, there are only two things we need to consider. They are the size of the state space $\{S_j\}$ and how to generate each possible solution according to a certain order. If the state space $\{S_j\}$ of a problems is small enough that it is possible to check all solutions by a set of moving methods $\{M_j\}$, we can always make sure to get all global optimal solutions by exaustive search within enough runtime. That is to say, the exaustive search is not applicable to the problem with a large size or even a middle size of state space which is related with the given runtime and the speed of computer. Even if the fastest supercomputer in the world is used, the problem solved by the exaustive search is still quite limited. The biggest advantage of exaustive search is its simplicity. As one embodiment of exaustive search to find global optimal solutions, breadth-first search (BFS) is used to the state space $\{S_j\}$ with graph structure. WFS means the algorithm exploits all adjacent vertices of a vertex reached in each step. Another embodiment is the opposite of BFS. That is depth first search (DFS) for the state space $\{S_j\}$ with tree structure. DFS exploits the sub-tree of the current node from root to leaf as deep as possible in each step. In comparison with WFS and DFS, steepest-first search (SFS) always follows the steepest move to its neighbor each time. However, SFS belongs to local search, instead of exaustive search.

For linear optimization problems, the iterative search with a systematic search procedure is used to get a global optimum. Simplex method is used to solve linear programming problems whose objectives and all constraints are linear. The only important thing we need to consider is the basic feasible solutions, because there is a proved fundamental theorem says that some optimal solution of a linear programming problem is also a basic feasible solution. The so-called basic feasible solution is the feasible solution for which at least n-m of the variables $\{x_i\} = \{x_1, x_2, \ldots, x_n\}$ are zero, where n is the total number of variables and m is the total number of independent

constaints. That means we can always find an optimum in the set of basic feasible solutions if the optima exist, though a linear programming problem can have a lot of optima but not all of optima belong to the set of basic feasible solutions. However, even if *n* and *m* are relatively small, there are C_n^m different ways to satisfy at least *n-m* of the variables $\{x_1, x_2, \ldots, x_n\}$ are zero, so it is still difficult to search from infeasible solutions to basic feasible solutions by considering all possibilities. The simplex method, which is proposed to overcome the mentioned difficulty by G. B. Dantzig in 1948, is a systematic search process which proceeds step by step from one to another basic feadible solution in the way of the objective function always increasing its value until reaching an optimum. The remaining point is how to find a basic feasible solution to start the simplex method. Besides, the extra slimination step should be done in the case of a degenerate feasible solution which is such a feasible solution that more than *n-m*, i.e. the usual number, of the variables $\{x_1, x_2, \ldots, x_n\}$ are zero.

2.1.2 Local Search and Greedy Algorithm

The basic method to get a local optimum is local search. For a normal local search, the initial solution is randomly produced as the current solution in the state space $\{S_j\}$. A candidate is one of solutions produced by a moving method in $\{M_j\}$ randomly operating the current solution. A cost function $\{C_j\}$ is used to compare the current solution with the candidate. If the cost is improved, the candidate replaces the current solution. Otherwise, the current solution keeps no change. Local search terminates if no better candidate is found when reaching a local optimum as the final output. However, some local optimal solutions are not always near-optimal solutions or not good solutions. The final solution quality and the computational time are closely related with the selected moving method in $\{M_j\}$ and the initial solution S_I.

Greedy algorithm is a local search with a heuristic rule. The initial solution of greedy algorithm is randomly produced in the state space $\{S_j\}$ as the current solution. A neighbor is one of solutions produced by a moving method in $\{M_j\}$ operating the current solution. A cost function $\{C_j\}$ is used to check all neighbors of the current

solution. If the cost is improved, the best neighbor replaces the current solution. Otherwise, greedy algorithm terminates and output the current local optimum. The difference between greedy algorithm and local search is as follows. The local search improves its solution each move, while the greedy algorithm selects the best moves after checking all possible moves each time. The greedy algorithm adds a heuristic rule: always follow the best move to get the maximum gain each time. It is such a simple rule that it is attractive for problem solvers to apply it conveniently. In fact, the normal local search is a random ascent hill-climbing method, while the greedy algorithm is actually a steepest ascent hill-climbing method or steepest-first search (SFS).

2.1.3 Random Search and its Extensions

The basic method to solve a general problem is trial-and-error method, i.e. random search, which is actually a special local search using random moves each time. Any local optimum of random search is also a global optimum because random moves reaches anywhere in the state space $\{S_j\}$ in theory. The trial-and-error method includes two stages. The trial stage is to try a solution randomly. The error stage is to evaluate the solution and return to the trial stage. The output is the best of all evaluated solutions. The biggest advantage of trial-and-error method is its generality. It could be used to solve any problem, even if the problem is a black box in which the inside information is totally unknown.

There are many extended methods which are similar with trial-and-error method in form. One example is divide-and-conquer method which is used to divide a complex problem to a series of subproblems which are easier to be conquered, and then the subproblems are merged into the original problem to get the final answer. It is effective only if the cost of the divide-conquer-merge process is smaller than the cost of solving the original problem directly. Another example is branch-and-bound method which is used to reduce the state space $\{S_j\}$ by removing the uninteresting branch whose cost does not satisfy the upper-bound or lower-bound condition of the

cost of the current solution. The important point to use the branch-and-bound method is to find an effective way to calculate the upper-bound or lower-bound cost of solutions and branches. Besides, the Focus-and-relay loop is also an improvement of trial-and-error method in the section of similar optimum search which is proposed in this research.

2.2 Modern Heuristic Algorithms

In practice, heuristics are used to solve general problems by adding heuristic rules according to past sucessful experience. The heuristic rules are key factors to be balanced between local search and random search. A strong heursitc rule may lead to the loss of optimal solutions, while a weak heursitc rule may increase computational time. Since most heuristic rules and ideas are intuitive or empirical, the final solution would be approximately global optimum or acceptably local optimum in most cases. When there is no exact algorithm to solve a complex problem, heuristic algorithm is one of the most effective choices to get a practical solution.

It is well-known that stochastic methods reach near-optimal solution instead of local optimum. They are suitable for NP-hard problem solving but it might take much computational time. Comparing with stochastic approaches, local search does not take much computational time but it always reaches a local optimum which might be not good enough. In fact, most of stochastic methods are based on the tradeoff between random search and local search. On the one hand, local search is often incapable of nonlinear optimization problem solving. In that case, random search might be required. On the other hand, undirected random search is extremely inefficient to solve the problem with large solution space. Stochastic methods which are based on directed random search are widely used to modern heuristics such as simulated annealing, genetic algorithm, etc.

2.2.1 Simulated Annealing

Simulated annealing (SA) [11-13] is a directed random search with the heuristic rules from an annealing process in physical system. Based on the theory of statistical mechanics and the analogy between solid annealing and optimization problem, S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi proposed SA algorithm in 1983 [11]. The annealing is to heat up a solid with a very high temperature and then to cool it down slowly until it reaches or approaches its minimum energy state. Each state of solid represents a feasible solution of problem. The energy of the state is the value of cost function to evaluate the solution. The basic state with the lowest energy corresponds to the optimal solution with the best value of cost function. As shown in Table 2, it is an analogous relationship between simulated annealing algorithm for optimization problem and metal annealing process in physical system.

Table 2: Analogous relationship between simulated annealing algorithm for optimization problem and metal annealing process in physical system

Optimization problem	**Physical system**
Objective	Minimum energy
Solution space	State space
Feasible solution	State
Optimal solution	Basic state
Parameter control	Temperature control
Scheduling	Cooling
Local search	Quenching
Cost	Energy

In details, SA is a stochastic algorithm with iterative improvement. Each iterative step consists of changing current solution to a new solution, named a move to neighborhood. The acceptance probability of new solutions depends on the current temperature, which is scheduled from the highest temperature to the lowest temperature. Let $\hat{S}$ be the solution space with neighborhood structure. For any

solution S belongs to $\hat{S}$, we define the cost $C(S)$. The flow chart of standard SA is as follows. The initial solution is randomly produced. The parameters of temperature scheduling include the starting temperature T_0, the ending temperature T_e. A solution is updated by using a moving method. The new solution is evaluated by cost function and then it is compared to the old solution. The new solution is accepted with the probability $P = exp[-\Delta C/T]$, which depends on the cost difference ΔC and the current temperature T. If rejected, the current solution goes back to the old one. The terminal condition is reaching the lowest temperature T_e.

A non-optimal solution S is defined by local optimum if it cannot reach better solution by moving to any neighboring solution S′. That is to say, for any neighbor solution (S′) of local optimal solution (S), the inequality $C(S) \leq C(S')$ is always satisfied. The depth $D(S)$ of local optimal solution S is defined by $Max[C(S')-C(S)]$. The maximum depth of local optimal solutions in $\hat{S}$ is denoted by d. Let S_i be i^{th} solution explored by simulated annealing, and T_i be the temperature when S_i is explored. Let C_{opt} be the minimum cost. The equality $lim_{i\to\infty} C(S_i) = C_{opt}$ is satisfied with the following conditions: (1) The solution space $\hat{S}$ is finite and irreducible; (2) There exists an equilibrium distribution for the transition probability matrix; (3) $T_i \geq T_{i+1}$ and $T_i > 0$ for all i; (4) $lim_{i\to\infty} T_i = 0$; (5) $\sum_{i:\in(0,\infty)} [exp(-d/T_i)] = \infty$.

2.2.2 Genetic Algorithm

Table 3: Analogous relationship between genetic algorithm for optimization problem and evolution process in biological system

Optimization problem	**Biological system**
Optimal solutions	The fittest individual
A set of solutions	Individuals in a population
Cost function	Fitness function
Coding of solution	Chromosome
Operators	Crossover, Mutation, etc.

Genetic algorithm (GA) [14] is a directed random search with heuristic rules of natural selection and mechanism of evolution. One of the most famous heuristic rules is “survival of the fittest” to select the next generation. In GA, as shown in Table 3, the evolution of a population of individuals gradually improves the fitness from one generation to anoter generation by genetic operators including selection, crossover and mutation, etc. The theoretical study of GA started with the cellular machine in 1950s. The formal GA with application to combinatorial optimization is proposed in a published book named as “Adaptation in Natural and Artificial Systems” by John Henry Holland in 1975. Besides, one of the greatest advantages of GA is its intuitive image. That is, GA does not use much knowledge about the specific features of optimization problems to deal with the problem directly.

For implementing GA, there are several important things we need to consider as follows. (1) How to represent the problem to be sovled. Different methods of representation have a great impact on the performance of GA in term of computational time and accuracy. Two typical representations include the binary coding schemes and a vector coding using integers or real numbers in which each integer or real number represent a single parameter. (2) How to create the initial populaton. One way to form the population is randomly produced. Another way is using the prioro knowledge of the given problem to get converge within less runtime. (3) How to design the set of genetic operators. A set of genetic operators include selection, crossover and mutation. Wherein, the selection operator could be propotional selection named “roulette wheel” or ranking-based selection. Especially, the crossover operator which creates two new individuals from two existing individuals is considered the basic feature that makes GA different from other methods. Unlike crossover, the mutation is a monadic operator in which the individual is randomly reversed accoding to a specified rate to avoid premature convergence. Of course, it is not necessary to use all above operators. The design of operators depends on the problem and its representation. Besides, the genetic operators may also include reproduction, inversion, etc. (4) How to control the parameters. The main parameters of GA include the number of individuals in a

pipilation, crossover rate and mutation rate. A large number of individuals in a pipilation take more runtime, but the probability of convergence to an optimum is higher. A low crossover rate decreases the speed of convergence, but a high crossover rate tends to high satuation around one solution. A higher mutation rate leads to higher diversity but also lower stability. A lower mutation rate leads to lower probability of convergence to a global optimum. (5) How to evaluate the fitness. The fitness mainly connects the algorithm with the problem. The complexity of fitness function to evaluate the quality of solution depends on the optimization problem in which the solution performs the desired objectives and satisfies the constraints.

2.2.3 Other Emerging Heuristics

As we know, most heuristic algorhtms are based on random search or local search with self-discovery features and heuristic rules. Tabu search is a typical local search based algorhtm which is proposed by Fred W. Glover and Hansen independently in 1986. It is an iterative improvement method in which the search history is memorized to avoid the duplication of the searched areas by marking as tabu. The tabu rule drives the search to explore new search space and keep the diversity of searched solutions, so the tabu search has the global search ability. The process to overcome the shortcoming of local optimality is using a cost function which chooses the highest cost each iterative step. The current solution moves to the best admissible solution in the neighborhood according to the cost function which selects the moves of the most improvement or the least deterioration. It is possible to return to old solution at a previously visited path, and this might cause a cycling search. To avoid this problem, the tabu list is used to store the search history. And the forbidding stratety is designed to update the tabu list. Since the tabu length is limited, the duplications are not entirely avoided. Furthermore, some special rules such as aspiration criteria are added to break the tabu restriction by accepting super solutions. The biggest difference between simulated annealing and tabu search is the way of the acceptance or rejection of a new solution. The former is based on a probability of

temperature scheduling, while the latter is based on search history.

In fact, tabu search is an extension of local search with several heuristic strategies as follows. (1) Forbidding strategy. It controls what enters the tabu list to avoid cycling problems by forbidding certain moves. In the ideal case, the tabu list must store all previoudly visited solutions to check if the solutions are revisisted, but it requires too much memory. An alternative way is not visiting the solutions already visitied during the last T iterations, where the T is the length or size of tabu list. The probability of cycling is high with a small T, while the state space may not be completedly explored with a too large T. The tabu list is normally using a first-in-first-out structure. (2) Aspiration strategy. It is used to make a tabu solution free if the solution is with sufficient quality. Usually tabu restriction has a role in the constraint of the state space. As a special case of the constraint, a tabu solution is assumed aceeptable if an aspriration criterion applies regardless of the tabu status. It could be very significant to achieve the best performance by appropriately using the aspiration criteria including time-independent aspiration, time-dependent aspiration, objective aspiration, search diretion aspiration, influence aspiration, etc. (3) Freeing strategy. It controls what exists the tabu list and when exists. A solution is considered admissible if its attributes are not a tabu or if it has passed the aspiration criteria test. (4) Learning strategy. It collects information which is used to direct the subsequent searches during a search run. It consists of intermediate and long-term strategies with memory functions. The learning strategy seeks new solutions that exhibit good features by recording the features of a selected number of moves. The memory functions of learning strategy provide the function of internsification. (5) Short-term strategy. It manages the interplay among the above strategies, so it is also called as overall strategy. In short, tabu search assumes that a solution with better cost has a higher probability of leading to a better solution in a fewer number of iterations.

Variable neighborhood search (VNS) [17] is an algorithm which is based on local search and random search using the idea of exploiting systematical change of neighborhood structure within a series of local searches. A basic VNS with k neighborhood structures starts with initialization which gets an initial soltion S_0,

selects a set of neighhorhood structures N_k (k=1, ..., k_{max}) and sets k = 1. The first step is shaking which generates a solution S_1 at random. The second step is local search which starts with S_1 to find a local optimum S_2. The third step is the setting of value k and goes back to the first step. If S_2 is better than the incumbent, then let S be S_2 and let k be 1. Otherwise, it sets $k = k+1$ until $k = k_{max}$ and then returns to k=1 again as a cycle. In the whole search process, VNS has one stage for improvement, and it generates the restarted solutions at random. VNS with k neighborhood structures explores increasingly distant neighbors of the incumbent solution within many independent local searches.

Particle swarm optimization (PSO) [16] is an intelligent evolutionary algorithm which is proposed by James Kennedy and Russell Eberhart in 1995. It is based on swarm competition and collaboration between particles to find near-optimal solutions for optimization problem. It is inspired by the group behavior of birds foraging, migration and clustering as particle behaviors. All particles which are randomly initialized have a fitness function which determines the speed, direction and distance of particles. To develop PSO algorithm, three heuristic rules should be simplely followed. (1) To avoid colliding with the neighbors. (2) To match the speed of neighbors in a group move. (3) To move by iteration toward the objective of group and toward the center of group. The particles follow two interdependent tracks to update themselves in each iterative step. One track is to find other particles which are adapting each other. Another track is to find the common objective.

Neural network (NN) [15] is a parallel distributed information processing model which simulates the mechanism of human brain. NN has three major features as follows. It has the capability of parallel computing. It has the ability of learning from examples to adapt new environment. It tolerates the noise, fault or even damage of components. NN consists of a number of elements which are the model of biological neuron. An element has one output and several inputs which could be the input signals of external devices, the output of other elements or its own output as a loop. The elements connect each other by links with weights. The weight of link represents the strenghs of connection which can be considered as the long-term memory. Generally

speaking, NN is divided into two types of structures. The first type is feedforward network in which the neurons are grouped into layers and connected from one layer to the next layer. The connected layers are not within the same layer. Signals flow from the input layer to the output layer by unidirectional connections. The second type is recurrent network in which the outputs of some neurons are connected to the same neuron or to the neurons in former layers as feedbacks. Signals flow more flexibly in forward or backward directions. When an optimization problem is solved by NN, it is designed to optimize the energy function of network corresponding to the cost function of problem.

2.3 Applications to Design Optimizaiton

Design optimization [21-30] is one of the most important applications with regards to search methods especially high-performance heuristics. The first part of this section is about general product design to be optimized by efficient search methods. A large-scale product can be regarded as an integrated system which consists of a great number of subsystems, modules or components. The second part introduces general system design and physical design to be improved by efficient search methods. After sophisticated analysis and integration, an entire system is multi-layered from top to bottom. The remaining parts of this section are as follows. 2D/3D IC physical design optimization is introduced as a typical application. Especially, 2D/3D packing and placement optimization problems are formulated as two typical NP-hard problems with single objective and multiple objectives. In short, the major point of this section is how to get optimal design for real products by using search method as an effective approach to design optimization.

2.3.1 Product, System and Physical Design Optimization

The final goal of product design is to get an optimal design [21-30] in order to satisfy the requirements of customer and the specification of product as well as possible. An optimal design [21-30] means the overall best design of an entire system rather than the best design of each subsystem. It comes to the issue of computational efficiency by using search methods. All we can do is to approach the global optimal design as close as possible within a given time. The global optimal design is always the best of local optimal designs which is not necessary the best design of every single subsystem. If the runtime is long enough, it is possible to get each local optimal design by iterative improvement with an appropriate initial solution of optimization problem. Furthermore, each initial solution can be gotten by random generation in theory. It means that we can always approach the global optimal design by searching more local optimal designs.

The foundation of product design is the classification of design specialties as relatively independent technical fields which are used to analyze a real product and decompose a complex system to subsystems. A typical decomposition is from entire system to hardware subsystem and software subsystem. In fact, a general product design can be divided into three categories: software system design, hardware system design and integrated system design. Wherein, an integrated system may include general software, hardware and firmware. The firmware is a kind of special software which connects general software and hardware. The definitions of general software and hardware are as follows. Software is any abstract thing which has information. Hardware is any concrete thing which has energy. A more specific classification of product design is based on the unit system and the prefix system as follows. Firstly, there are some common units, such as time (s), frequency (Hz), energy (J) and power (W), which can be used to most of technical fields. For example, software also requires the above common units because it always needs a signal carrier with energy. Secondly, the prefix system (k, M, G,…) includes binary and decimal systems. The binary system (2^{10}, 2^{20}, 2^{30}, …) is normally used to software, while the decimal system (10^3, 10^6, 10^9, …) is used to hardware. Thirdly, the unit system is used to classify more detailed technical fields as follow. (1) Electronic units including current

(A), voltage (V), capacitance (F), resistance (Ω), inductance (H), magnetic flux (Wb), etc. (2) Mechanical units including length (m), mass (kg), force (N), pressure (Pa), etc. (3) Optical units including luminous intensity (cd), luminous flux (lm), luminosity (lx), etc. (4) Thermal units including Kelvin temperature (K), Celsius temperature (°C), etc. (5) Other units such as plane angle (rad), solid angle (sr), and so on.

The elements of product design [21-30] which is optimized by search methods include design variables, design objectives, design constraints, input and output. The design variables are all design parameters which are possible to be adjusted. The design objectives are the specification of an ideal product which satisfies all requirements of customer. The design constraints are all realistic rules which are related with design objectives. The input is all known relevant information which is related with design variables. The output is the final values of design variables and design objectives after optimization. The similarity and difference between mathematical optimization and design optimization are as follows. Just like mathematical optimization, two key elements of design optimization are objectives and constraints. If the objectives can be evaluated by a function, it is called cost function. If all constraints can be satisfied by a given solution, it is called feasible solution. For most product design, however, there is an interchanging relationship between objectives and constraints. Unlike mathematical optimization, most objectives of design optimization are flexible constraints, and most constraints of design optimization are fixed objectives in practice. For mathematical optimization, all objectives could be evaluated by a cost function under the constraints of a set of equations or inequalities. For design optimization, an objective could be a pure state which has no corresponding cost function to be evaluated. A mathematical optimization problem could be with no constraints, and the most important thing of the problem is its objectives. A design optimization problem could be with no objectives. The most important thing of a design optimization problem is its constraints and how to search a feasible solution which satisfies all required constraints. Even if there are many objectives of a design optimization problem, they are just relatively unimportant constraints which could be flexibly adjusted from the

customer's point of view. Besides, most of design optimization problems have several conflicting objectives with unknown weights. Pareto Improvement of one objective without worsening any other objectives should be the safest way to improve a design optimization problem in practice.

The concept of systematic product design starts with Richard Buckminster Fuller who describes it as "applying the principles of science to solving the problems of humanity". That is so-called design science, design methodologies or interdisciplinary design principles. It is an abstract of various designs in different technical fields such as software engineering, mechanical engineering, electronic engineering, etc. It is also possible to apply it to non-technical fields such as social science, environmental science, economics, politics, etc. For a large and complex system design, it is almost impossible to achieve the overall best design directly due to the high complexity of the entire system. The analysis and integration of a large system and its subsytems are normally necessary. The system analysis includes the study of system functionality, the interface between system and ourside world, the interconnection among subsystems, the hierarchy of system and subsystems, etc. By system analysis, all subsystems, modules and components are small enough to be realized. The system integration is completed by a compatible interconnection among subsystems rather than a comprehensive cover of an entire system according to a great number of successful design experiences. The value of interconnection is judged by the new features or additional functionality of the whole system which does not exist in the simple sum of its subsystems.

A basic model of system analysis and integration is a process of decomposing system to subsystems and combining subsystems into an entire system by interconnection, as shown in Figure 4, just like the assembly of a number of modules. It is named as the horizontal analysis and integration of a system with interconnected structure. Hierarchical structure of a large system is produced by repeated horizontal analysis and integration. As shown in Figure 5, the whole system consists of multiple layers from top to bottom. It is named as the vertical analysis and integration. The top layer is normally user interface. From the enginner's point of view, the top layer is the

most abstract model, which the bottom layer is the most concrete components. The bottom layer generally consists of physical modules and components, so it is named by physical design which can be extended to any bottom-layer design of a system with hierarchical structure. The hierarchical system could be integrated system, hardware system, or even software system.

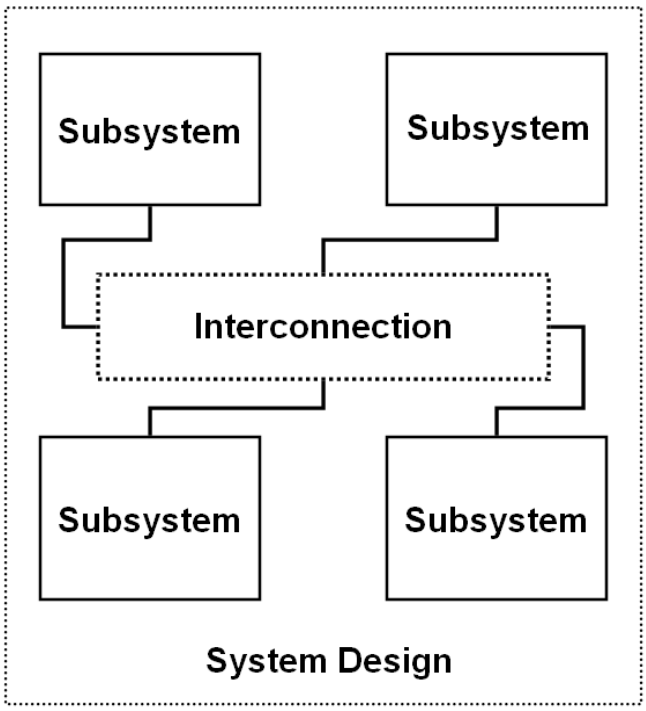

Figure 4: Interconnected structure of a large system: the horizontal analysis and integration of system and its subsystems as modules

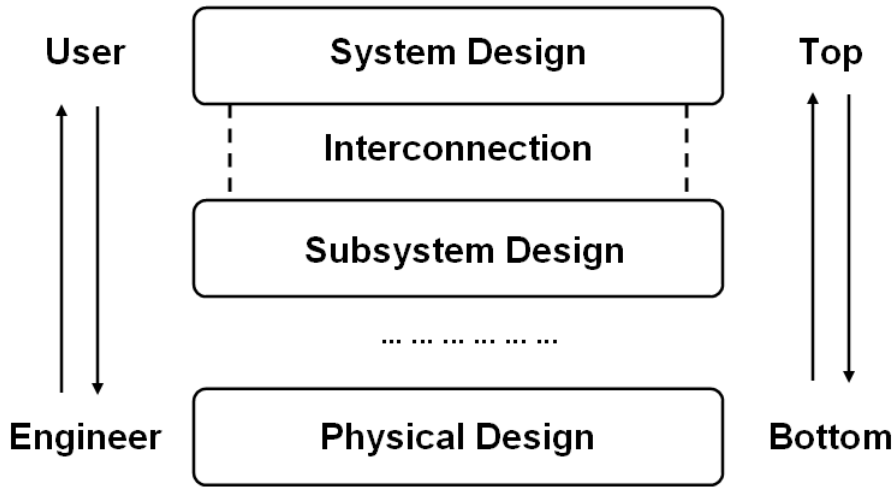

Figure 5: Hierarchical structure of a large system: the vertical analysis and integration produced by repeated horizontal analysis and integration.

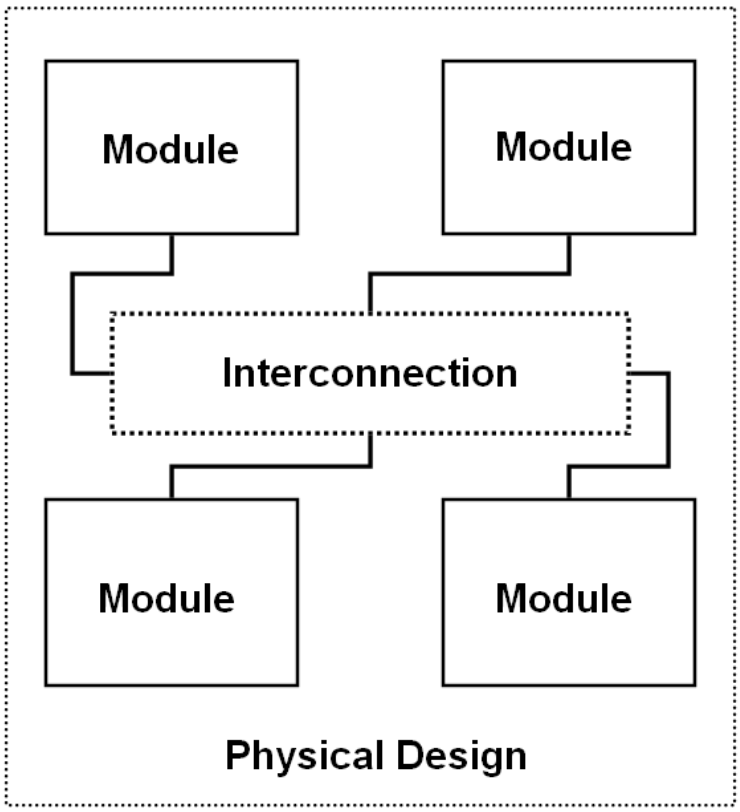

Figure 6: Physical design of the bottom layer: the assembly and interconnection of a number of modules which could be any concrete entities

A general physical design on bottom layer is the assembly and interconnection of a number of modules, as shown in Figure 6, which could be any concrete entities. As a typical multi-layer system, the Open Systems Interconnection (OSI) is a model of a communication system which is proposed by international organization for standardization (ISO). The OSI model puts all connected computers into a framework of World Wide Web. In the OSI model, an entire networking system is divided into seven layers. A layer serves the layer above it and is served by the layer below it. The top layer is application layer which is closest to the end user. The sixth layer is presentation layer that establishes context and provides a mapping between different syntax and semantics of top-layer entities. The fifth layer is session layer which establishes, manages and terminates the connections between computers. The fourth layer is transport layer which provides the reliable data transfer services to the upper

layers. The third layer is network layer which provides the functional and procedural means of transferring variable length data sequences from one node to another connected to the same network. The second layer is data link layer which provides a reliable link between two directly connected nodes. The bottom layer is physical layer which has the following functions. It defines the physical specifications of connection. It defines the relationship between a device and a physical transmission medium such as a copper or fiber optical cable. It defines the protocol to establish, manage and terminate a connection between two directly connected nodes and more.

2.3.2 2D/3D IC Physical Design Optimizaiton

As a typical modern product, 2D/3D integrated cicuit (IC) drives the revolution of computer and communication technologies which are the basic parts of information revolution. IC technology has been and continues to be evolving from small scale 2D IC with a few transistors to very large scale integation (VLSI) chips with millions of transistors and even 3D IC chips with billions of transistors in future. IC chip consists of a great number of electronic components which are built on one or more silicon wafer. In general, the design process of 2D/3D IC chip includes specification, architecture, functional design, logic design, circuit design, physical design, trial fabrication, testing and mass production. In details, the specification is the technical description of various performances according to the requirements of customers. As a high-level representation of an entire system, the specification is normally a compromise among technology, user requirement, market and economics. Many factors such as performance, functionality, physical dimensions, design techniques and fabrication should be considered in this step. The architecture is an overall system design on the top layer which includes many decisions about the system and its subsystems. The outcome of architecture is a micro-architectural specification which is a textual description. The early estimates of performance can be done according to the description of architectural design. The function design is to about the functionality of each subsystem or module. Main functional units and the interconnect

requirements between units are identified. The performance of each unit is estimated. The outcome of functional design is a timing diagram or other relationships between functional units. The logic design is normally about the register transfer level description to realize the logical relationships among modules and components. The description consists of Boolean expressions and timing information which can be used in the simulation of system design and the verification of its correctness. The circuit design is to develop a circuit representation to realize the interconnect relationships among all electronic circuit elements such as cells, macros, gates, transistors, etc, as a detailed circuit diagram. The Boolean expressions of logical design are converted into a circuit representation by taking into consideration the requirements of power and speed. The circuit simulation can be used to verify the correctiness and timing of each component. The physical design is to realize a geometric description of 2D/3D IC circuit which is specific shapes in several layers. The trial fabrication, testing and mass production are the process of converting the data of physical design into IC chips as real products to satisfy all performances in the specification.

During IC physical design, the circuit representation is converted into geometric representation. The geometric representation of IC circuit is called a layout which consists of a set of planar geometric shapes. The layout is created by converting each logic component such as cells, macros, gates, transistors into geometric shapes which perform the logic function of the corresponding component. The design files are used to describe IC layout. The interconnections between different components are also expressed as geometric patterns typically lines in multiple layers. The pattern files are used to produce patterns, i.e, masks, which is converted from the design files by using optical pattern generator. The exact details of the layout also depend on design rules, which are guidelines based on the limitations of the fabrication process and the electrical properties of materials. By a sequence of photo-lithograhoic steps during fabrication, these masks are used to pattern a silicon wafer and to produce all components and lines step by step. Layout optimizations and verifications are performed during IC physical design process.

2D/3D IC physical design optimization is a complex process which is broken

down into many sub-steps such as partitioning, floorplanning, placement, routing, and so on. Let us take partitioning optimization and routing optimization as two examples. The input of partitioning optimization is circuit diagram including schematics and net list. Since an IC chip may contain millions of transistors, it may not be possible to optimize the entire system of a large circuit in only one step. The circuit is normally partitioned into sub-circuits as modules or blocks. The partitioning optimization should consider many factors which could include the number of modules, the size of modules and the number of interconnection between modules. In a large circuit, the partitioning process may also be hierarchical. Each module may be partitioned recuisively into many smaller sub-modules. The output of partitioning optimization is a set of modules and the interconnections between modules. With regard to routing optimization, it is required to complete the interconnections between modules according to the specified netlist. The objectives of optimization depend on the type of IC chips. For general purpose chips, it is sufficient to minimize the total wire length of all interconnections. For high performance chips, it is more important to optimize the maximal delay of each interconnect wire to meet the timing requirements. The conventional approach to routing optimization is divided into two phases. The first phase called global routing generates a loose route for each net. It assigns a list of routing regions to each net without specifying the geometric layout wires. The second phase called detailed routing finds the actual geometric layout of each net within the assigned routing regions. Most problems in routing optimization are computationally hard, so researchers focus on heuristics instead of exact search methods.

2.3.3 Packing and Placement

As shown in Figure 7, the applications and benchmarks of search methods in this research mainly focus on floorplanning and placement as two typical NP-hard problems. The floorplanning in 2D/3D IC physical design [31-49] can be regarded as a packing problem with flexible or fixed shapes of modules. The size and shape of each module or block can be estimated according to the output of circuit partitioning.

The modules for which the dimensions are yet to be determined are called flexible modules, while the modules for which the dimensions are determined are called fixed modules. In the case of 2D IC physical design, the input of floorplanning is from the output of circuit partitioning which includes a set of modules or blocks, the area of each module, possible shapes of each module, the number of terminals for each module and the netlist. The output of floorplanning is the location of each module on the layout surface as well as the location of pins on the boundary of the modules which can be determined by the rotation of modules.

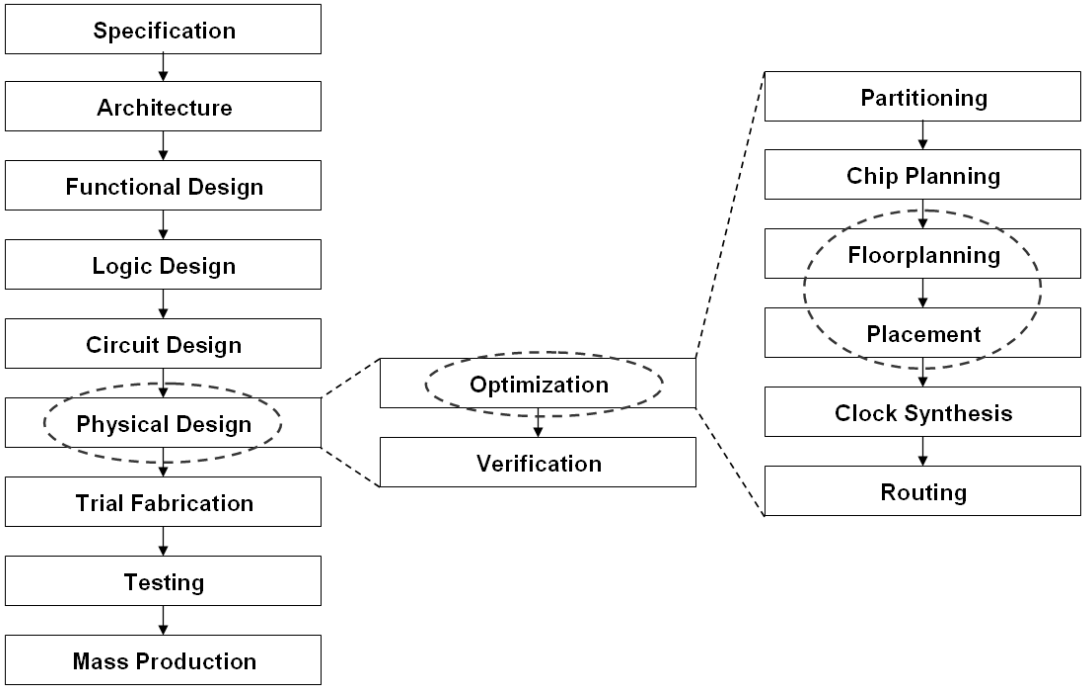

Figure 7: The floorplanning and placement of physical design optimization in the design process of 2D/3D integrated cicuit (IC).

The placement [40-49] occurs at three different levels: system level, board level and chip level. Let us take the chip level as an example. The dimensions of each module can be estimated after the circuit partitioning in 2D/3D IC physical design and the netlist specifying the connections is available between modules. The iterative improvement of placement is made until the layout has the best performance and minimum dimensions to satisfy design specifications. Besides, some space between modules is left empty to allow interconnections between modules. In fact, the quality

of placement will not be evident until the interconnection has been completed. The routing may not be possible in the space left by placement without a fine estimate of interconnection and the required routing space. The overall performance depends heavily on placement due to the fact that few moves can be done to improve the routing and the performance, once the location and rotation of each module is fixed.

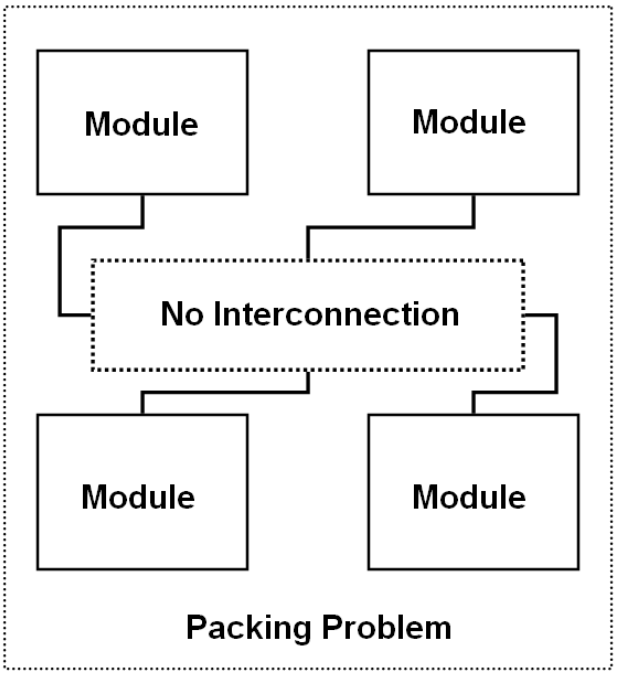

Figure 8: A packing optimization problem in physical design.

Packing or floorplanning optimization [31-49] is a typical single objective NP-hard problem. In general, the input of a packing problem in physical design is a set of modules with fixed dimensions. The objective is to minimize the overall bounding dimensions which contain all modules. The constraint is no overlops between modules. The output is the location and the rotation of modules with the optimized objecive. The interconnection between modules is not considered in the packing optimization, as shown in Figure 8.

2D rectangular placement problem is regarded as 2D rectangular packing with interconnect optimization. As shown in Figure 9, the formulation is as follow. Let $M = \{m_1, m_2, ..., m_n\}$ denote the modules or blocks to be placed, where n is the number of modules. Each m_i, where $1 \le i \le n$, has height h_i and width w_i. Let (x_i, y_i, r_i) of module

m_i be the location and rotation on 2D orthogonal coordinate system, where (x_i, y_i) means the coordinates of the south-west corner of module m_i, and r_i represents the rotation (0, 1) of m_i on xy-plane. r_i =1 is the normal state of modules with vertical height and horizontal width, while $r_{z\text{-}i}$ =0 is rotated by 90 degree.

As one of the objectives, the packing area is defined by the minimum bounding rectangle including all modules. For the interconnect networks of placement, let $N = \{n_1, n_2, ..., n_l\}$ be the set of interconnect nets between modules, where l is total net number. Let len_i denote the estimated wire length of each net n_i, $1 \le i \le l$. Let P_i denote the estimated dynamic power, i.e. the interconnect power of net n_i. In short, the input is a set of modules $M = \{m_1, m_2, ..., m_n\}$ with height and width $\{(h_1, w_1), (h_2, w_2),... , (h_n, w_n)\}$ and a net list $N = \{n_1, n_2, ... , n_l\}$. The constraint is no overlap between m_i and m_j, where $i \neq j$. The output is a set of location and rotation of modules $\{(x_1, y_1, r_1), (x_2, y_2, r_2),..., (x_n, y_n, r_n)\}$ such that:

1. Minimize the power consumption $\sum_{1 \le i \le l} P_i$.
2. Minimize the maximal delay $\underset{1 \le i \le l}{Max}[Len_i]$.
3. Minimize the bounding area.

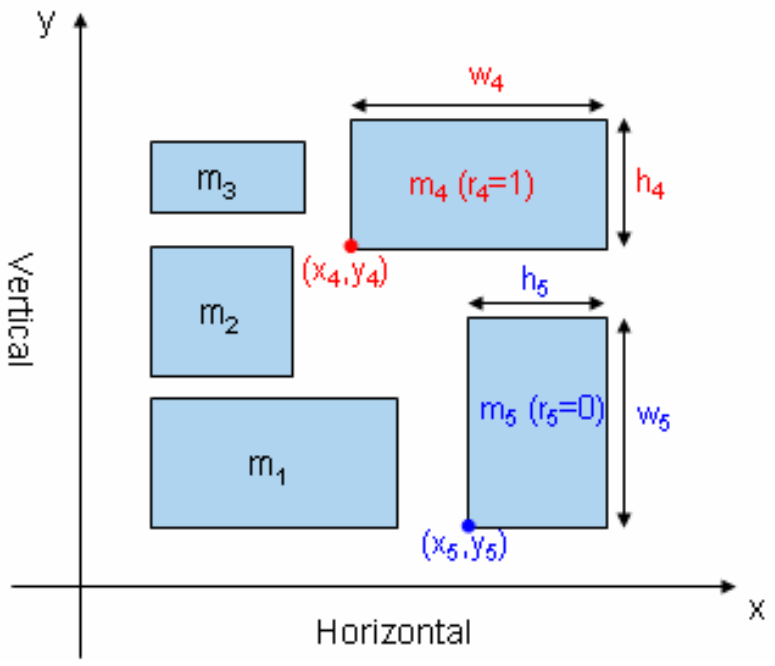

Figure 9: 2D rectangular placement as a special case of 3D placement

The conventional notation of “Above, Below, Left and Right (ABLR)” is changed to “North, South, West and East (NSWE)” to avoid the conflict of “Below (B)” and “Bottom (B)” when 2D placement is extended to 3D placement by adding “Top and Bottom (TB)”. A general 3D rectangular topology is represented by using sequence quintuple (SQ) representation, but the solution space of SQ is quite large. SQ representation is simplified to sequence triple (ST) representation, which can be decoded to a relatively simple 3D topology. In the case of 2D topology, sequence pair (SP), which can be simplified from ST, is used to represent a general 2D rectangular topology. As an example, the coding of Figure 9 is gotten by using the coding-decoding transition method in Figure 10 and Figure11, which are based on North-South and West-East relation corresponding to the order of modules in sequence pair.

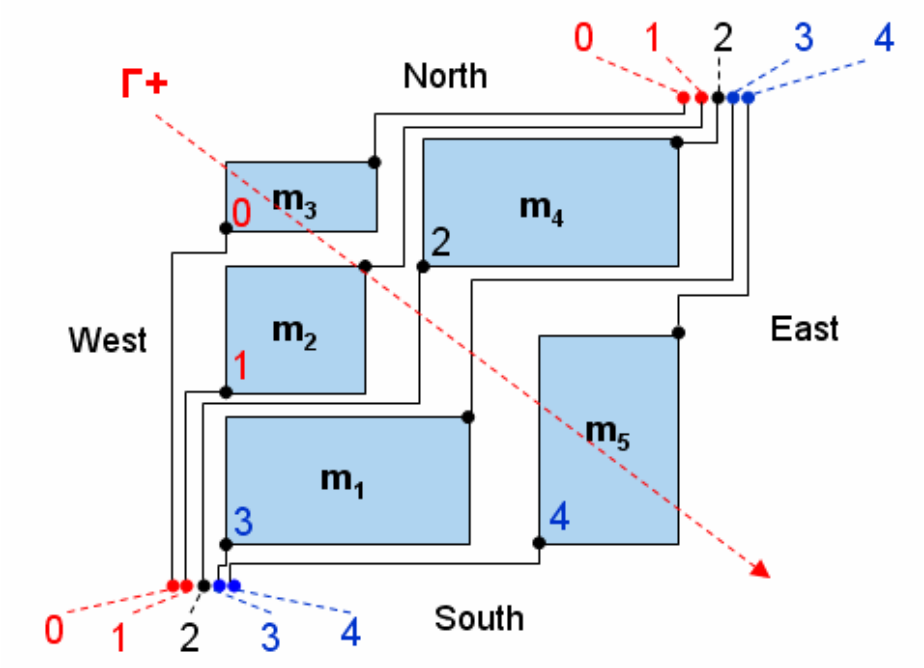

Figure 10: Coding-decoding transition method to get Γ+ (m_3,m_2,m_4,m_1,m_5) using the West-South and East-North corners of module to connect with those corners of layout

In order to get a positive sequence Γ+, each West-South corner of module connects to the West-South corner of the whole layout, and each East-North corner of

module connects to the East-North corner of the whole layout without any intersection. We get a sequence (m_3,m_2,m_4,m_1,m_5) corresponding to (0,1,2,3,4) from left side (i.e. two red points in Figure 10) to right side (i.e. two blue points in Figure 10). The positive sequence Γ+ is changed from the West-North corner to the East-South corner, i.e. from 0 to n-1 in Γ+, where n is the total number of modules. By using this coding method, we get the whole sequence Γ+ as shown in Figure 10, which is Γ+ (m_3,m_2,m_4,m_1,m_5). Similarly, in order to get Γ-, each West-North corner of module connects to the West-North corner of the whole layout, and each East-South corner of module connects to the East-South corner of the whole layout without any intersection. As shown in Figure 11, we can get (m_1,m_2, m_5,m_3,m_4) as Γ- using the same method to get Γ+. The negative sequence Γ- is changed from the West-South corner to the East-North corner, i.e. from 0 to n-1 in Γ-. So far, we get the coding of a general 2D topology based on North-South and West-East relation corresponding to the order of modules in sequence pair.

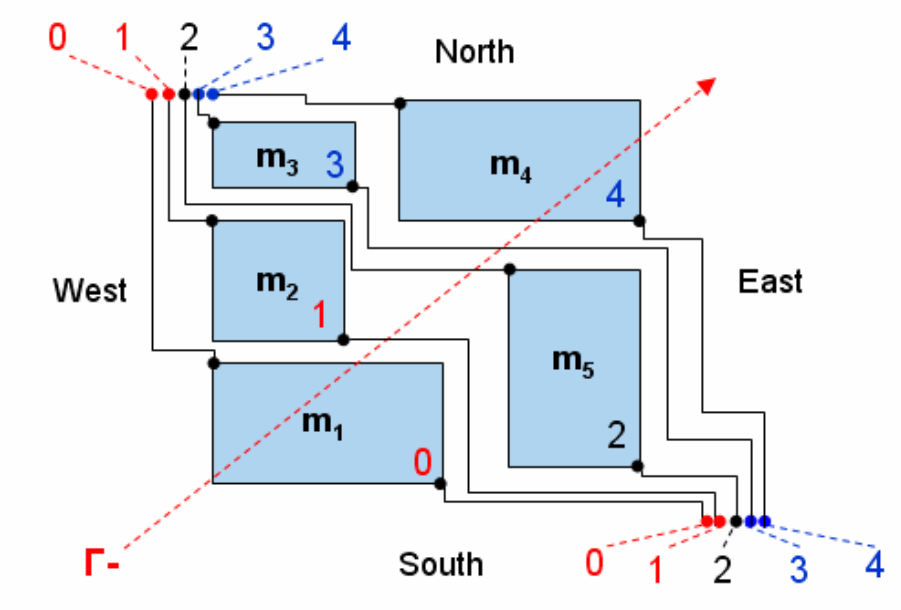

Figure 11: Coding-decoding transition method to get Γ- (m_1,m_2, m_5,m_3,m_4) using the West-North and East-South corners of module to connect with those corners of layout

Two sequences (Γ^1, Γ^2) generate finite solution space which includes at least one

optimal solution of 2D topology for area optimization by decoding the representation. Let Γ^i =(Γ^i[0], Γ^i [1], ..., Γ^i [n-1]) be one of two sequences, where i is 1 or 2.Let $F^i(m_j)$ be the order of m_j in sequence Γ^i. 2D topology is regarded as a set of the relations of relative location between modules, i.e. "North-South" and "West-East" (NS- and WE-) relations. Let (m_i N m_j) and (m_i W m_j) denote NS- and WE-relations. SP defines (m_i W m_j) when

$$F^1(m_i) < F^1(m_j) \text{ and } F^2(m_i) < F^2(m_j).$$

It defines (m_i N m_j) when

$$F^1(m_i) < F^1(m_j) \text{ and } F^2(m_i) > F^2(m_j).$$

The rule of symmetry to be followed is that (m_i N m_j) is the same relation as (m_j S m_i). That is to say, the topology should be reversely decoded if the order of labeling is reversed. For a given packing with n modules, the solution space is $(n!)^2$. If the rotation of the module is not fixed, then the solution space will increase to $(n!)^2 2^n$. Normally SP is decoded in time complexity $O(n^2)$. The decoding time is improved to $O(nlogn)$, further to $O(nloglogn)$ using FAST-SP, and even to $O(n)$ using Selected SP.

2.4 Summary

Search methods including exhaustive search, local search, random search and heuristics are widely used to approach optimal solutions of NP-hard problems. With the fast advancement of product design, it is increasingly hard to satisfy the high-performance requirements of computation due to short life cycle, large design scale and high complexity. In typical search methods, an adjacent solution is randomly obtained by a moving method. If the adjacent solution satisfies some given conditions, then the current solution is replaced with it. Otherwise, it is rejected, and the current solution is kept. For example, local search rejects an adjacent solution if it is worse than the current solution. It stops when it regards the current solution as a local optimum. However, the final solution obtained by the iterative improvement method is often far worse than a global optimum which is the best of all local optimal solutions.

In order to get a better solution, various modern heuristic algorithms such as simulated annealing have been introduced. Simulated annealing accepts a nonimproved adjacent solution with some probability according to temperature scheduling to escape from a local optimum. However, the efficiency is not good enough since an obtained local optimum is independent with others. A local optimal solution which is similar to an obtained local optimum would not be explored. It is expected that a better local optimum exists which is similar to an obtained local optimum. Besides, packing and placement problems of 2D/3D IC physical design optimization are introduced as the benchmarks to evaluate the computational efficiency of improved search methods in this research.

3. Techniques for High Performance

This chapter, which includes three sections, is mainly on the techniques to improve heuristics [1-19] and design optimization [21-30]. In the first section, the technique of moving method is discussed. Based on the crossover of genetic algorithm, a special crossover between the current solution and the local optimum is proposed. Diverse moving methods including group rotation and group exchange are discussed. In the second section, the technique of adaptive selection is discussed. Move selection ratio and adaptive guide are introduced to improve existing algorithms by selecting moving method adaptively. In the third section, similar optimum search is proposed to solve the problem with a great number of local optimal solutions. The move-improve loop and focus-and-relay loop are introduced. Based on the combination of high-performance techniques, adaptive guide with special crossover and similar optimum search with diverse moves are introduced as the basic ideas of new proposed heuristics.

3.1 Technique of Moving Method

3.1.1 Special Crossover

The crossover operator of genetic algorithm (GA) [14] is designed to reuse the searched solutions by creating two new children from two existing parents. There are many different ways to design crossover operators. Let us take the sequential configuration of solution as an example. The basic crossover operator is shown in Figure 12. Two solutions are randomly selected from a given population as parent 1 and parent 2. A random cutting point is produced, and two sequences of parents are divided into four parts. Two new solutions named as child 1 and child 2 are produced by swapping four parts of sequences. Just like the basic crossover operator, the cycle

crossover operator is shown in Figure 13.

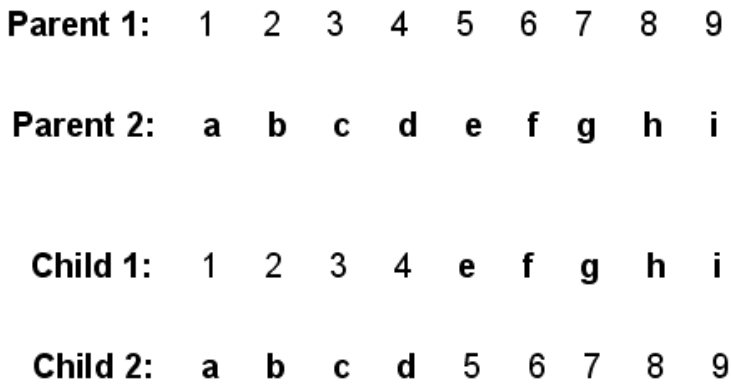

Figure 12: Basic crossover operator of genetic algorithm (GA)

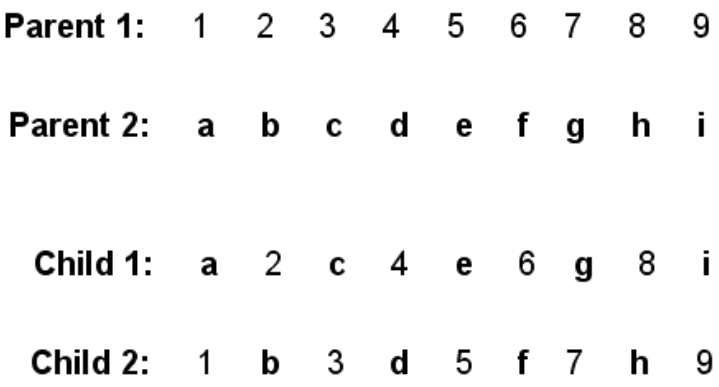

Figure 13: Cycle crossover operator of genetic algorithm (GA)

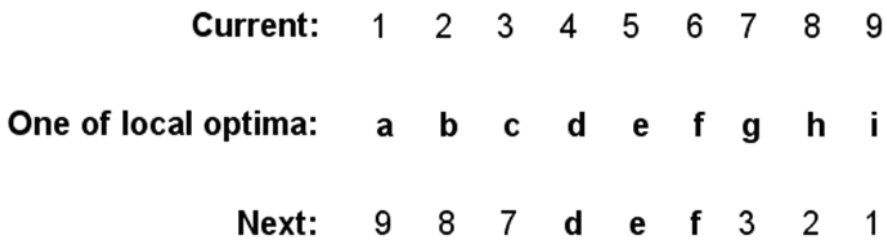

Figure 14: The reversed crossover with one of local optima as a special moving method for adaptive simulated annealing (ASA)

As shown in Figure 14, it is the reversed crossover as a special moving method for adaptive simulated annealing (ASA). The margin of the next solution is from the

reversed margin of the current solution, and the center of the next solution is from the center of the local optimal solution. To realize the special crossover operator for ASA, there are three difficulties to be overcome. Firstly, simulated annealing (SA) based algorithms have only one state at a time. Unlike the population-based crossover operators of GA, the special crossover is designed to generate a new solution from the current solution and one of searched local optimal solutions. Even if two given solutions, we can get a new solution are same by reversing the local optimal solution. Secondly, the acceptance probability is unstable. Because one of two states is from the local optimal solution, it has more chance to get a better solution by crossover. As a result, the acceptance probability of crossover is improved. Thirdly, it is hard to approach a local optimum by using crossover due to its big move at a time. An adaptive guide to adjust the selection probability of crossover is needed to improve the acceptance probability and the final convergence. As shown in Figure 15, it is the random crossover as a special moving method for ASA. The next solution is inherited from the current solution and one of local optimal solutions according to a random code. The interesting point is the traditional SA with crossover is inefficient, while the proposed ASA with crossover is efficient according to the experimental data of this research.

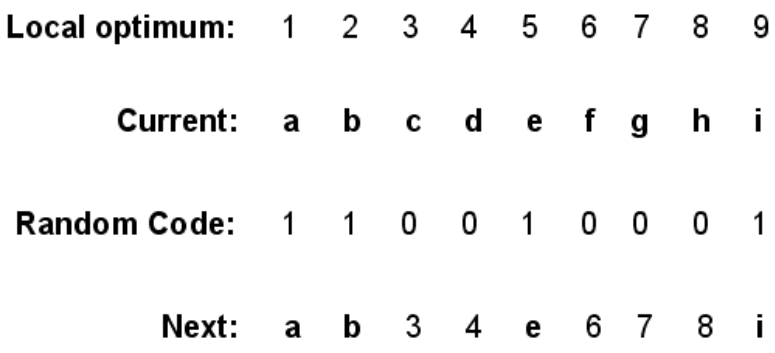

Figure 15: The random crossover with one of local optima as another special moving method for adaptive simulated annealing (ASA)

3.1.2 Diverse Moving Methods

Diverse moving methods could be group rotation, group exchange, group insertion, and so on. They are typically designed as follows. The group rotation is repeated rotations of randomly selected modules with a given number. The group exchange is repeated exchanges of randomly selected pairs of modules with a given number. The group insertion is repeated insertions of randomly selected pairs of modules in a selected sequence with a given number.

In general, a local optimum defined by one set of moving methods is not necessarily a local optimum defined by another set of moving methods. Solution space, moving method and cost function are three elements of a search method. In the solution space, the adjacency between solutions is defined by moving methods that modify one solution to another solution. A local optimum is a solution whose cost is not worse than any adjacent solutions. Whether a solution is a local optimum or not depends on the adjacency defined by moving methods. A global optimum is the local optimum defined by random moves.

Local optimum search is based on the definitions of state, cost and move. Let the solution space $\{S_j\}$ be the set of all states corresponding to the set of cost $\{C_j\}$, and let $M_i(S_j)$ define a new state which is gotten by a given moving method M_i operating a given state S_j as the following.

$$S_k = M_i(S_j)$$

where S_k is a new state after the operation of M_i to S_j. Let $\Phi = \{M_i\}$ be the set of moving method which consists of the set of focusing moving method $\Phi^f = \{M_i^f\}$, the set of rough moving method $\Phi^r = \{M_i^r\}$ and the set of relay $\Phi^e = \{M_i^e\}$ as the following.

$$\Phi = \Phi^f \bigcup \Phi^r \bigcup \Phi^e$$

Let the set of basic moving method $\Phi^b = \{M_i^b\}$ be a subset of Φ^f as the following.

$$\Phi^b \subseteq \Phi^f$$

A state S_m which is gotten by a path of moving methods $\varphi = (M_1, M_2, \ldots, M_i, \ldots, M_{|\varphi|})$ sequentially operating S_n is defined as the following.

$$S_m = \varphi(S_n) = M_{|\varphi|}(\cdots M_i(\cdots M_2(M_1(S_n))\cdots)\cdots)$$

where M_i is a moving method in Φ, and |φ| is the length of φ. In short, the state, cost and move are three elements of local optimum search to design moving methods.

The state-cost distribution is based on the basic moving methods, the distance of state and difference of cost. If only the basic moving methods in Φ^b are used, the set of a path from one state S_n to another state S_m is defined as the following.

$$\varphi^b(S_n, S_m) = \{\varphi = (M_1, M_2, ..., M_i, ...M_{|\varphi|}),$$
$$M_i \in \Phi^b \ \ | S_m = \varphi(S_n)\}$$

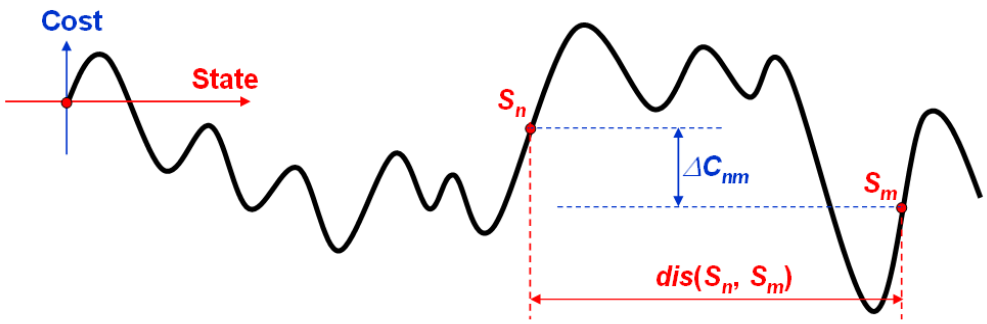

Figure 16: Distance of state and difference of cost on state-cost distribution

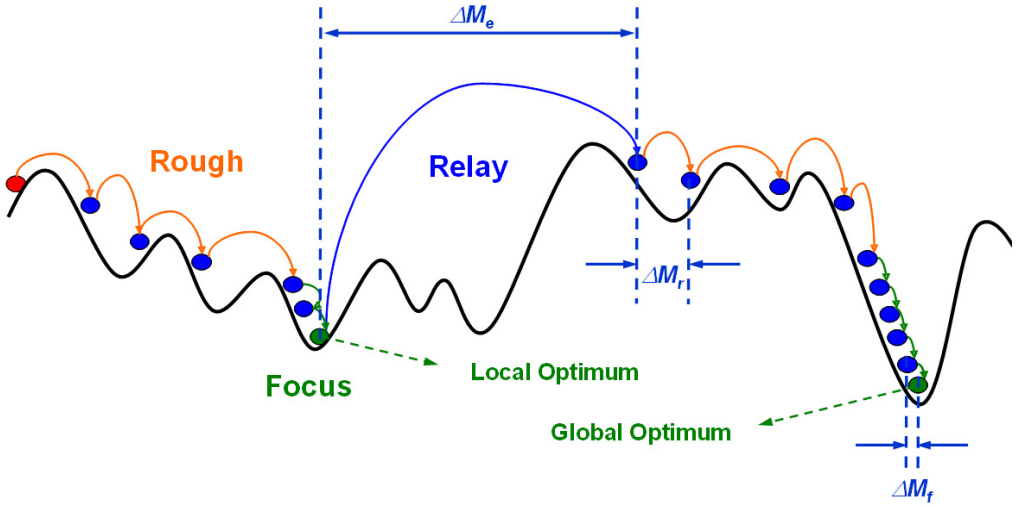

Figure 17: Rough search, focusing search and relay

Then, the distance between S_n and S_m is defined as the following, as shown in Figure 16.

$$dis(S_n, S_m) = \min\{|\varphi| \ \ | \varphi \text{ in } \varphi^b(S_n, S_m)\}$$

Let ΔC_{nm} be the difference of cost between two states S_m and S_n as the following, as shown in Figure 17.

$$\Delta C_{nm} = \left| C_n - C_m \right|$$

where C_n and C_m are the cost of S_n and S_m.

To classify moving methods, the upper-bound value of the set of distances $\{dis(S_k, S_j)\}$ is defined as the following.

$$\Delta S_{\max} = \max(\{dis(S_k, S_j) \mid k \neq j\})$$

Let ΔM_f, ΔM_r or ΔM_e be the upper-bound value of the set of distances from S_k to $S_j = M_i(S_k)$ using moving methods M_i in Φ^f, Φ^r or Φ^e as follows.

$$\Delta M_f = \max(\{dis(S_k, M_i(S_k)) \mid M_i \in \Phi^f\})$$

$$\Delta M_r = \max(\{dis(S_k, M_i(S_k)) \mid M_i \in \Phi^r\})$$

$$\Delta M_e = \max(\{dis(S_k, M_i(S_k)) \mid M_i \in \Phi^e\})$$

To design the focusing moving methods, the rough moving methods and the relay, the following condition is recommended but not necessary to be satisfied as shown in Figure 17.

$$\Delta S_{\max} \geq \Delta M_e >> \Delta M_r >> \Delta M_f \geq 1$$

where the notation of ">>" means "much larger than".

3.2 Technique of Adaptive Selection

3.2.1 Move Selection Ratio

In order to select diverse moving methods, the move selection ratio is designed with different probabilities according to an adaptive guide. The selection ratio is a relative value, which is calculated by improvement ratio and the previous selection ratio. The initial probability is same for each moving method. The next selection ratio is adapted according to the current selection ratio and the improvement ratio. The total

probability 100% is always maintained.

3.2.2 Adaptive Guide

To realize the idea of guide, a series of probabilities are adjusted automatically to select moving methods. The frequency of improvement $f_k(j)$ of j^{th} moving method in k^{th} iteration loop is defined by

$$f_k(j) = \frac{t_{improve}(j)}{t_k(j)}$$

where $t_k(j)$ is the number of trials using j^{th} moving method in k^{th} iteration loop and $t_{improve}(j)$ is the number of improved trials among $t_k(j)$ trials. The amplitude of improvement $a_k(j)$ of j^{th} moving method in k^{th} iteration loop is defined by

$$a_k(j) = \sum_{i=1}^{t_k(j)} Max[0, -\Delta C_k(j,i)]$$

where $\Delta C_k(j, i)$ is the difference of cost at i^{th} trial of j-th method in k^{th} iteration loop. The improvement ratio $s_k(j)$ is defined by

$$s_k(j) = \frac{a_k(j) f_k(j)}{\sum_{j=0}^{l-1} a_k(j) f_k(j)}$$

where l is the number of moving methods. If no improvement happens during the iteration loop, the guide is not updated. The initial probability p_0 is same for each moving method. The next probability p_{k+1} is given by $(s_k+p_k)/2$ according to the current probability p_k and the improvement ratio s_k.

3.3 Technique of Similar Optimum Search

Let Λ be the set of solutions. Let $C(S)$ be the cost of solution S in Λ. A moving method modifies a solution to another solution. The solution obtained from solution S by moving method M is denoted by $M(S)$. Let $\varphi = (M_1, M_2, \ldots, M_{|\varphi|})$ be a sequence of moving methods. The length of φ which is the number of moving methods in φ is

denoted by $|\varphi|$. The solution obtained from solution S by sequence of moving methods φ is denoted by $\varphi(S) = M_{|\varphi|}(\ldots M_2(M_1(S))\ldots)$. A sequence φ of moving methods is said to be from solution S_i to solution S_j when $\varphi(S_i) = S_j$.

Let B be the set of moving methods. A moving method in B is said to be *basic* hereinafter. In this paper, a local optimal solution defined in terms of B is simply referred by local optimal solution in the following unless otherwise specified. A sequence of moving methods is said to be *basic* if it consists of basic moving methods. In this paper, the distance between solutions in solution space is defined by using basic sequences. The distance $\mathrm{dist}(S_i, S_j)$ from S_i to S_j is defined as the minimum length of basic sequences from S_i to S_j. That is, $\mathrm{dist}(S_i, S_j) = \min\{\ |\varphi|\ |\ \varphi \text{ in } P(S_i, S_j)\}$ where $P(S_i, S_j)$ is the set of basic sequences from solution S_i to solution S_j .

In most of search algorithms proposed so far, it is natural to consider that B consists of all moving method used in each search algorithm. There is no restriction on a moving method in B, but it is often assumed that a moving method M in B modifies a solution S slightly, and the difference between $C(S)$ and $C(M(S))$ is small. Moving methods used in SA are usually expected to satisfy this property.

3.3.1 Move-Improve Loop

The move-improve loop is a greedy approach which is used to reach a local optimum using any moving method, which could be basic moving method, focusing moving method, rough moving method or relay. As an iterative improvement process, the move-improve loop is a basic part of local search as well as most heuristics. The flow chart of move-improve loop is as shown in Figure 18. The input could be a random solution as a current solution. The current solution is updated as a candidate by moves. All solutions could be evaluated by a cost function. If the candidate is better than the current solution, it moves again using the same moving method. Otherwise, the candidate returns to the old solution. A moving method is selected for the next move which is used to update solution until a predetermined number of repeated nonimproved solutions. The output is a local optimum.

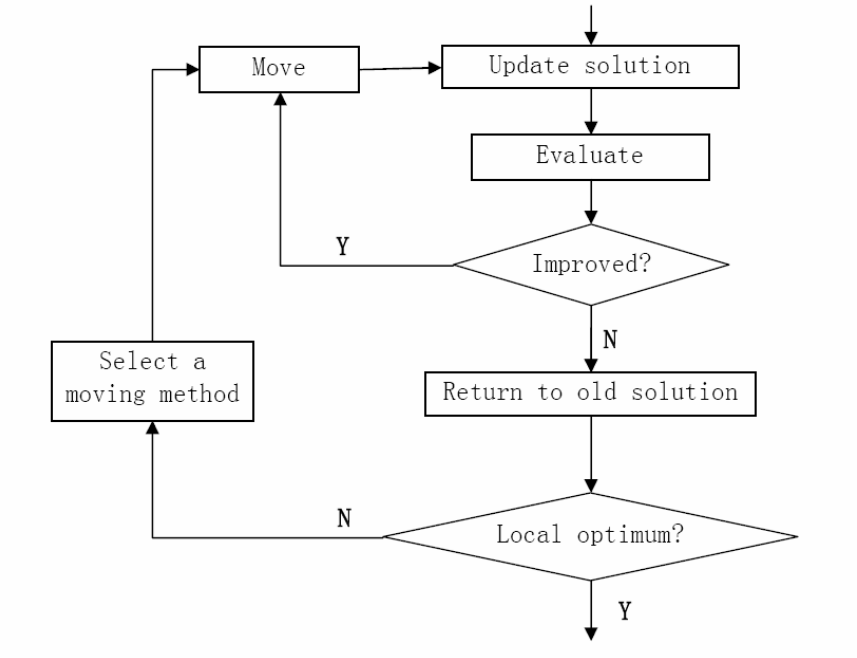

Figure 18: Search local optimum by move-improve loop.

3.3.2 Focus-and-Relay Loop

Focus-and-relay loop is the core of similar optimum search to approach a global optimum as shown in Figure 19. The focus stage is designed to search a local optimum each move-improve loop. The relay stage is designed to escape the current local optimum by randomly generating a part of the current solution. The iterative number of the focus-and-relay loop depends on the number of similar local optimal solutions. Based on the focus-and-relay loop, the similar optimum search with multi-stage improvement is designed as follows. It starts with initialization and rough search. Then it is changed to the focusing search after a given number of trials with no improvement. During focusing search, the solution is considered to be a local optimum if no improvement is obtained with a given number of trials, and be relayed to escape the local optimum in only one step. Then it is changed to rough search again until satisfying the terminal condition, in which all runners with a predetermined number are relayed.

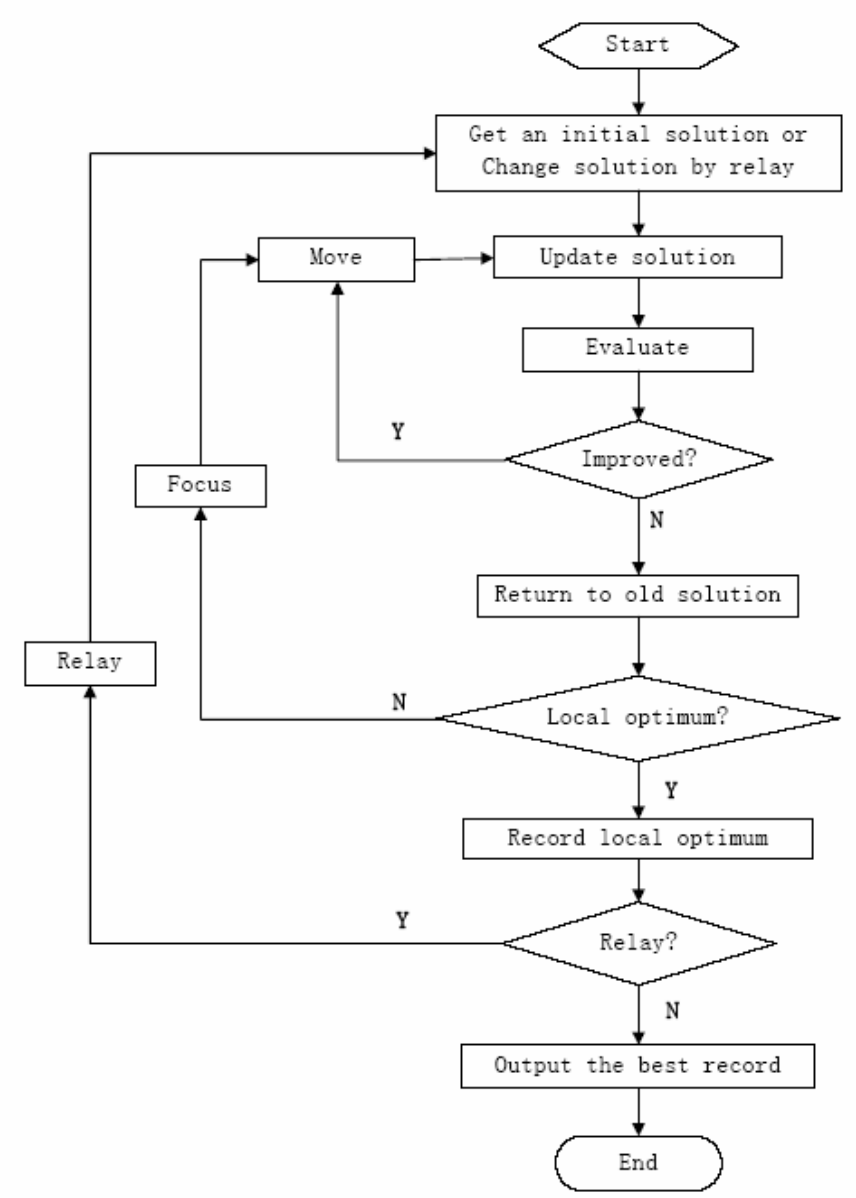

Figure 19: Focus-and-relay loop as the core of similar optimum search

The global ratio of relay (R_e) is the percentage of the new part within the whole state. If the relay is regarded as a moving method, the parameter R_e should satisfy the following equation.

$$R_e = \frac{\Delta M_e}{\Delta S_{\max}}$$

where ΔM_e is the upper-bound distance between two states using relay, ΔS_{max} is the upper bound of distance between any two states, and R_e is a value between 0 and 1 which is used to keep the balance between local part and random part. In general, no similarity lasts long unless there is diversity in it. The local part is about similarity,

while the random part is about diversity. We believe that no heuristic is general unless there are both simlaritiy and diversity in it.

3.4 Summary

Based on the discussion of search methods, high-performance techniques are introduced. Firstly, as the technique of moving method, a special crossover between the current solution and the local optimum is proposed. Diverse moving methods including group rotation and group exchange are discussed. Secondly, the technique of adaptive selection is discussed. Move selection ratio and adaptive guide are introduced to improve existing algorithms by selecting moving method adaptively. Thirdly, similar optimum search is proposed to solve the problem with a great number of local optimal solutions. The move-improve loop and focus-and-relay loop are introduced. Finally, based on the combination of the mentioned techniques, two new high-performance heuristics are proposed to improve the efficiency of problem solving in many different technical fields such as 2D/3D IC physical design optimization where existing search methods have failed to be efficient or effective.

4. Adaptive Simulated Annealing with Crossover

This chapter is a new improved heuristic algorithm [1-19] which consists of two sections to improve design optimization [21-30]. The first section introduces the basic idea to design adaptive simulated annealing with crossover (ASA_X). In this section, fast scheduling of simulated annealing, guide to select moving method, crossover and other moving methods are introduced. The second section is about parameter settings of the proposed ASA_X.

4.1 Adaptive Guide with Special Crossover

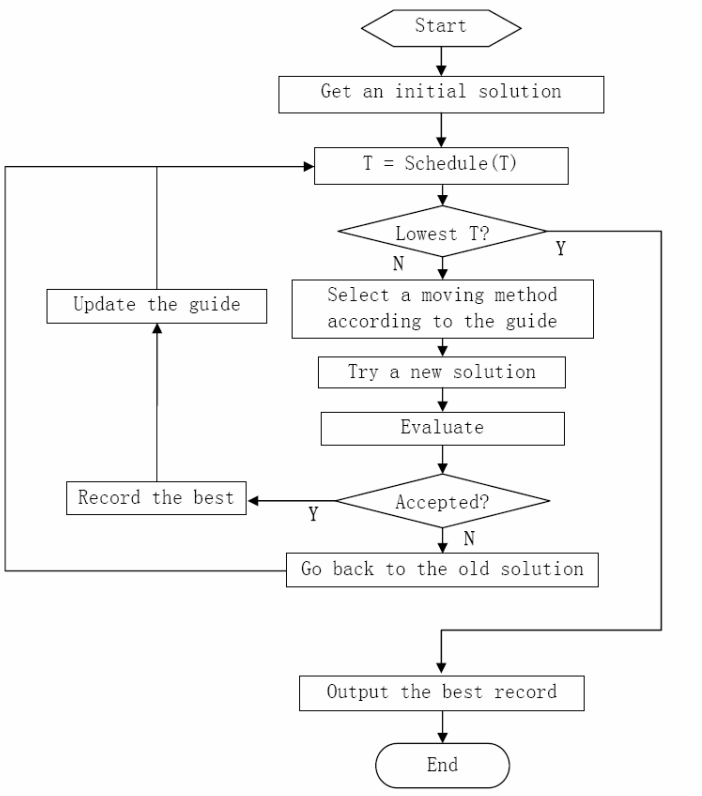

Figure 20: Flow chart of a new variation of adaptive simulated annealing (ASA)

Adaptive guide with special crossover is the basic idea to propose the new variation of adaptive simulated annealing (ASA_X). In the traditional ASA, the

parameters that control temperature and random step selection are adjusted according to search progress. The proposed ASA is a stochastic iterative improvement method with a guide which is designed to use the experience of past moves by using an adaptive guide to select moving methods incuding a specil crossover. The flow chart of new ASA is as shown in Figure 20. By using the guide, one of moving methods is selected according to adaptive probabilities whose initial values are same for each moving method. If the new solution is better than the current solution, the adaptive guide will be updated. When reaching a local optimal solution, the best record is updated. The output will be the latest best record. The interesting point is the traditional SA with crossover is inefficient, while the proposed ASA with crossover is efficient due to the guide with adaptive probabilities to select diverse moving methods.

4.1.1 Fast Scheduling

Simulated annealing (SA) is one of the most popular search methodes so far. SA finds near-optimal solutions for a general optimization problem by the anology of reaching minimum energy in metal annealing process with the distribution of system energies detemined by the Boltzmann probability. SA is a typical algorithm using the technique of scheduling. The temperature scheduling is designed to improve local search and random search.

The flow chart of SA is as shown in Figure 21. An initial solution is randomly produced. After temperature scheduling, a moving method is selected. The selection probabilities for different moving methods are same in this research. A new solution is tried by using the selected moving method. Then the new solution is evaluated and compared with the old solution. The solution might be accepted with a calculated acceptance probability $P = exp[-\Delta C/T]$ according to the evaluation and the current temperature. If the solution is improved, the new solution will be compared with the best-so-far solution to decide whether the best record is updated or not. If rejected, the current solution goes back to the old solution. It continues the next temperature

scheduling until reaching the lowest temperature T_e. The output will be the best record.

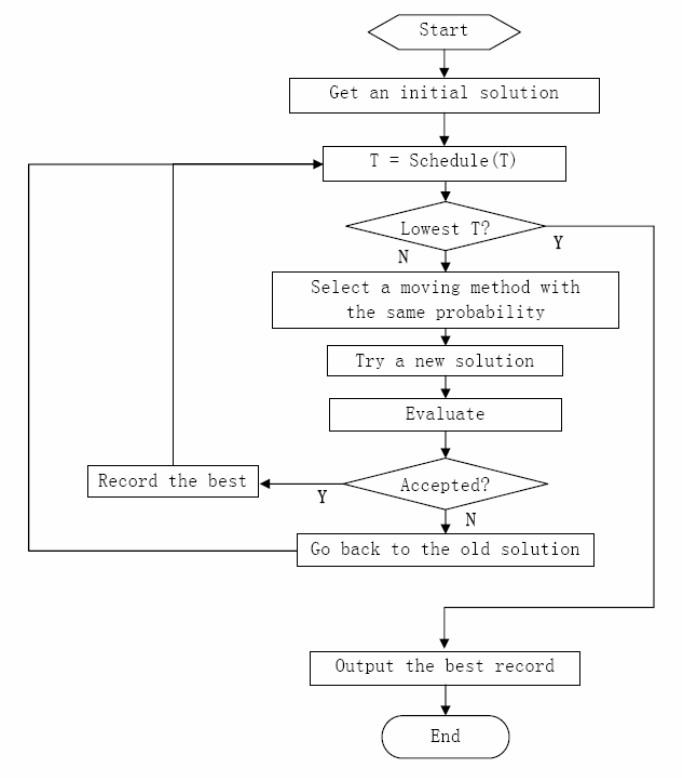

Figure 21: Flow chart of simulated annealing (SA)

In general, a moving method that may cause a drastic solution change is not used in SA. This feature limits the search ability of SA since the number of required iterations including rejection from a local optimum to another local optimum is large. It is normally inefficient to solve the problem with huge solution space. Based on two-stage scheduling, traditional SA is improved by classification of diverse moving methods. Based on the technique of adaptation, adaptive simulated annealing (ASA) is designed to select diverse moving methods adaptively. Based on the technique of search history, a special crossover operator is designed to improve adaptive simulated annealing further. The special crossover does not use the configuration of past solution as another kind of search history.

The key point of SA is the temperature scheduling. The parameters of a fast temperature scheduling include the starting temperature T_0, the ending temperature T_e,

a temperature coefficient and an inside loop number. The fast temperature scheduling is using an annealing for temperature decreasing exponentially according to $T_i = T_0 \exp(-ci^{1/D})$. This annealing is faster than fast Cauchy annealing, where $T = T_0 / i$, and much faster than Boltzmann annealing, where $T = T_0 \ln i$. If we assume $a = \exp(-c)$ and $q_i = i^{1/D}$, we will get the following equation.

$$T_i = T_0 \cdot a^{q_i}$$

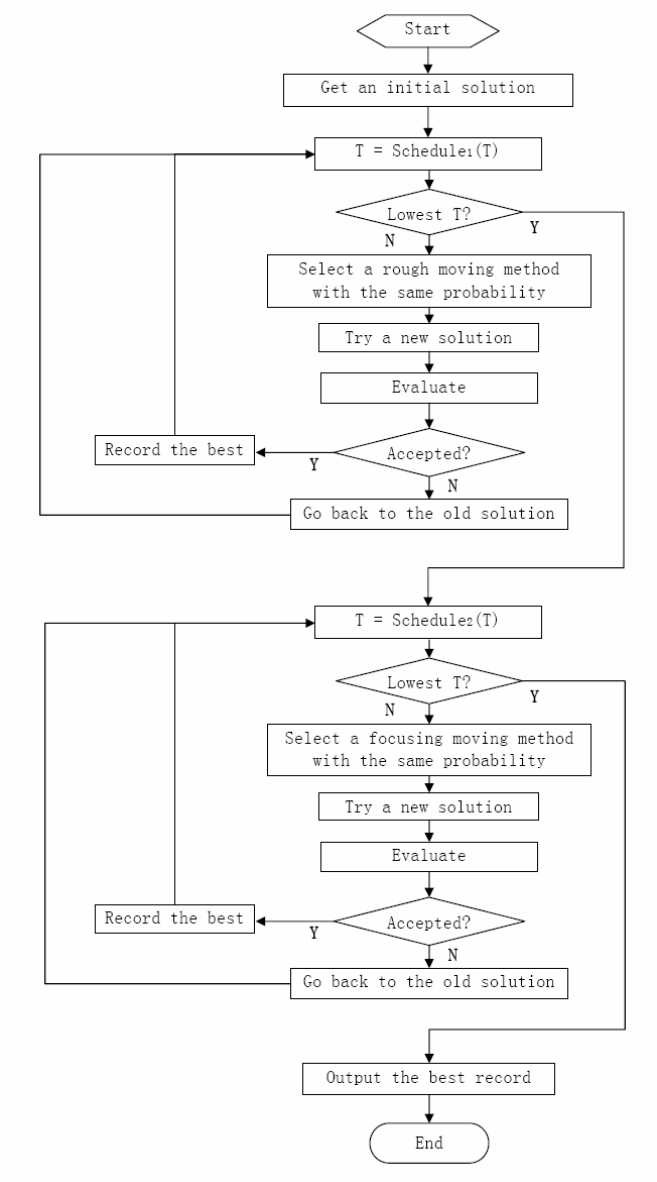

Figure 22: Flow chart of two-stage simulated annealing (TSA)

Let p be the number of repeating the same temperature, we get the following equation.

$$q_i = \left(\frac{i}{p} \right)^{1/D}$$

Let q_i be an integer and $D = 1$. The temperature scheduling is simplified to an easy-to-calculate version according to the scheduling $T_{i+1} = aT_i$ ($0<a<1$) with p times repeated inside loop, where i is the iterative step, T_i is the variable temperature at the i^{th} step, T_0 is the initial temperature when $i = 0$, p is an integer of the inside loop number, a is the temperature coefficient between 0 and 1, and a is normally near 1.

Two-stage fast scheduling can be used to improve simulated annealing by overcoming the shortcoming of limited moving methods. In two-stage simulated annealing (TSA), the moving method with small changes is labeled as focusing moving method, while the moving method with big changes is labeled as rough moving method. It starts with the first stage using the first temperature scheduling. After reaching the lowest temperature of the first scheduling, it is changed to the second stage using the second temperature scheduling, in which different parameters are used. Different moving methods are selected with the same probability during both stages in this research. The flow chart of TSA is controlled from the first stage to the second stage as shown in Figure 22. An initiation solution is randomly produced. The moving methods are divided into rough moving methods and focusing moving methods. TSA tries a new solution by using a rough moving method at the first stage and using a focusing moving method at the second stage. The solution is evaluated by the cost function (C). The solution is accepted with a probability $P = exp[-\Delta C/T]$, where T is the temperature, P is between 0 and 1. If $\Delta C < 0$, then $P = 1$. The parameters of temperature scheduling are denoted as follows. The initial value and the ending value of temperature are T_0 and T_e during the first stage. For the second stage, T'_0 and T'_e are used to replace T_0 and T_e. A temperature coefficient between 0 and 1 is set to control the speed of temperature reduction. An inside loop number is set to control the repeated moves for each T. When the new solution is better than the current best, the best record is updated.

To improve the computational performance of simulated annealing for packing optimization, adaptive simulated annealing with crossover (ASA_X) is proposed by

overcoming the shortcomings of simulated annealing (SA). In the traditional ASA, the parameters that control temperature scheduling and random step selection are adjusted according to search progress. In the proposed ASA, a guide with adaptive probabilities is used to automatically select moving methods, including crossover to improve its efficiency.

4.1.2 Guide to Select Moving Method

The probability (p_{k+1}) of guide to select a moving method is adaptive according to the former probability (p_k) of guide and the short-term improving speed $s_k(j) = a_k(j) \cdot f_k(j) / \sum a_k(j) \cdot f_k(j)$, where j means different moving methods, a_k is the relative amplitude of improvement, and f_k is the frequency of improvement. The frequency of improvement $f_k(j)$ of j^{th} moving method in k^{th} iteration loop is defined by $t_{improve}(j)/t_k(j)$, where $t_k(j)$ is the number of trials using j^{th} moving method in k^{th} iteration loop and $t_{improve}(j)$ is the number of improved trials among $t_k(j)$ trials. The amplitude of improvement $a_k(j)$ of j^{th} moving method in k^{th} iteration loop is defined by the maximum value between 0 and the difference of cost at i^{th} trial of j-th method in k^{th} iteration loop. We pick up $-\Delta C>0$ to calculate a_k, so the guide will not be updated if no improvement happens. We get p'_{k+1} given by $(p_k+s_k)/2$, and then calculate the new probability p_{k+1} given by $p'_{k+1}(j)/\sum p'_{k+1}(j)$ for each running method to keep the total probability 100%. For example, if the old probability for three running methods is given by $[p_k(1), p_k(2), p_k(3)]=[20\%, 30\%, 50\%]$, the relative amplitude of improvement is $[a_k(1), a_k(2), a_k(3)]=[60\%, 30\%, 10\%]$, and the frequency of improvement for each running methods is $[f_k(1), f_k(2), f_k(3)]=[50\%, 10\%, 90\%]$, then the new probability p_{k+1} will be $[p_{k+1}(1), p_{k+1}(2), p_{k+1}(3)]=[46\%, 18\%, 36\%]$.

4.1.3 Crossover Moving Method

A special crossover operator is designed as follows. Two sequences are derived

from the current solution and the best solution, and the remaining sequences are derived from the current solution. Two sequences (Γ^+, Γ^-) out of three sequences (Γ^1, Γ^2, Γ^3) or five sequences (Γ^1, Γ^2, Γ^3, Γ^4, Γ^5) are picked up randomly for 3D as shown in Figure 23. Let us denote the father as (Γ_f^+, Γ_f^-) which is selected from the current solution. The mother (Γ_m^+, Γ_m^-) is from the best solution so far. A number i is an integer randomly produced between 1 and $n/2$–1, where n is the number of modules. The child of SP (Γ_c^+, Γ_c^-) is given by $\Gamma_f^+[0, i] + \Gamma'^+_m + \Gamma_f^+[n–i–1, n–1]$ and $\Gamma_f^-[0, i] + \Gamma'^-_m + \Gamma_f^-[n–i–1, k–1]$, where Γ'^+_m and Γ'^-_m are the inverse of $\Gamma_m^+ – \Gamma_f^+[0, i] – \Gamma_f^+[n–i–1, n–1]$ and the inverse of $\Gamma_m^- – \Gamma_f^-[0, i] – \Gamma_f^-[n–i–1, k–1]$, respectively.

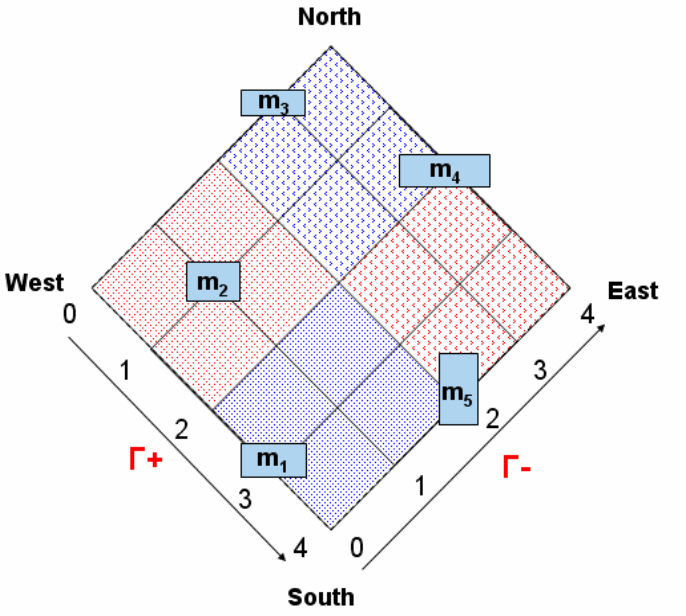

Figure 23: An example of two sequences Γ^+ and Γ^-, which are selected randomly from (Γ^1, Γ^2, Γ^3) or (Γ^1, Γ^2, Γ^3, Γ^4, Γ^5).

To make it clear, let us take an example to explain the crossover. As shown in Figure 24, the left layout is represented by $\Gamma^+=(m_3,m_4,m_2,m_1,m_5)$ and $\Gamma^-=(m_5,m_1,m_2,m_3,m_4)$ as the father. The right one is $\Gamma^+=(m_1,m_3,m_4,m_2,m_5)$ and $\Gamma^-=(m_3,m_4,m_1,m_5,m_2)$ as the mother. Assume that i = 1. Then, the child is the layout represented by $\Gamma^+=(m_3,m_2,m_4,m_1,m_5)$ and $\Gamma^-=(m_5,m_2,m_1,m_3,m_4)$ as the right layout of Figure 25, where $\Gamma^+=(m_3, ..., m_5)$ and $\Gamma^-=(m_5, ..., m_4)$ are from the father as the margin

of left picture of Figure 24, and Γ^{+}=(...,m_2,m_4,m_1,...) and Γ^{-}=(...,m_2,m_1,m_3,...) are from mother with an inverse order as the center of right picture of Figure 24.

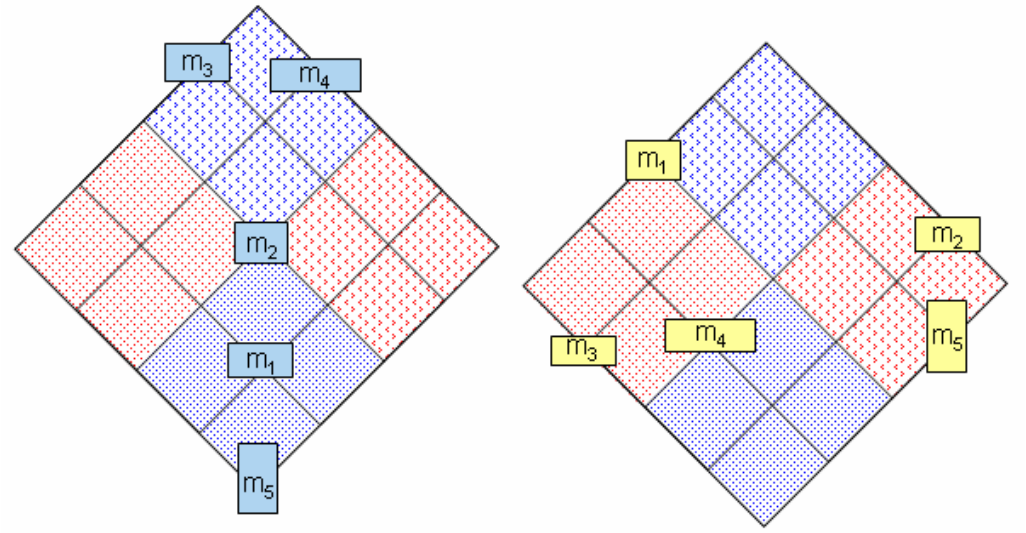

Figure 24: Crossover operator as a moving method: the current solution and the best-so-far solution (Father and Mother)

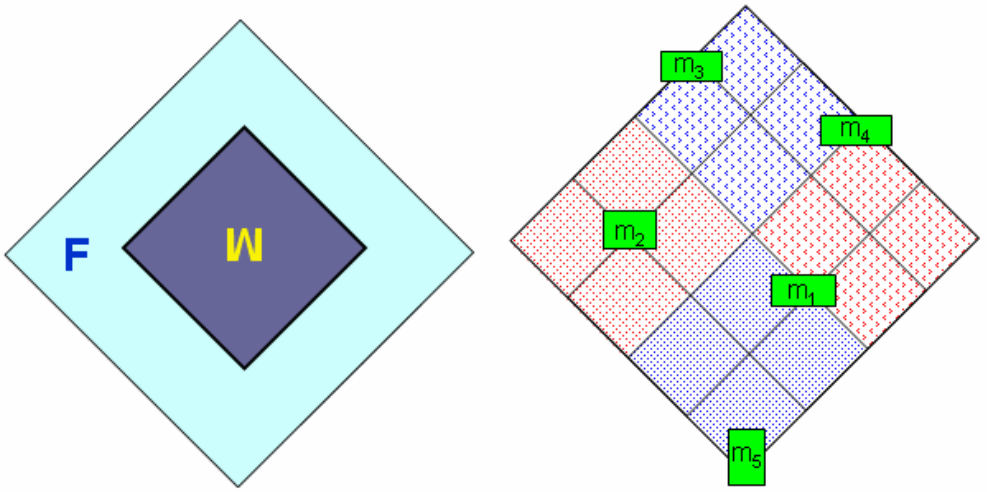

Figure 25: Crossover as a moving method: the next solution (Child)

4.1.4 Other Moving Methods

Three moving methods including rotation, exchange and insertion are defined as follows. Firstly, the rotation changes the orientation of a module. When a rotation is

applied to module m_i, r is changed to 1 - r, where r is one of ($r_{xy\text{-}i}$, $r_{yz\text{-}i}$, $r_{zx\text{-}i}$). Secondly, the exchange moving method exchanges the order of two modules in all sequences, e.g. in sequence pair (Γ^1, Γ^2), $F_1(m_i)$, $F_2(m_i)$, $F_1(m_j)$, and $F_2(m_j)$ are changed to $F_1(m_j)$, $F_2(m_j)$, $F_1(m_i)$, and $F_2(m_i)$, respectively. Thirdly, the insertion changes the order of a module in one sequence Γ^i. When an insertion is applied to module m in Γ^i, $F^i(m)$ is changed to another value, say j, and the orders of modules whose order is between $F^i(m)$ and j are shifted accordingly. For example, if the operation is to insert m_5 to the first position in Γ^- in Figure 26, the insertion leads to $F_-(m_1) = F_-(m_1) + 1$ and $F_-(m_2) = F_-(m_2) + 1$.

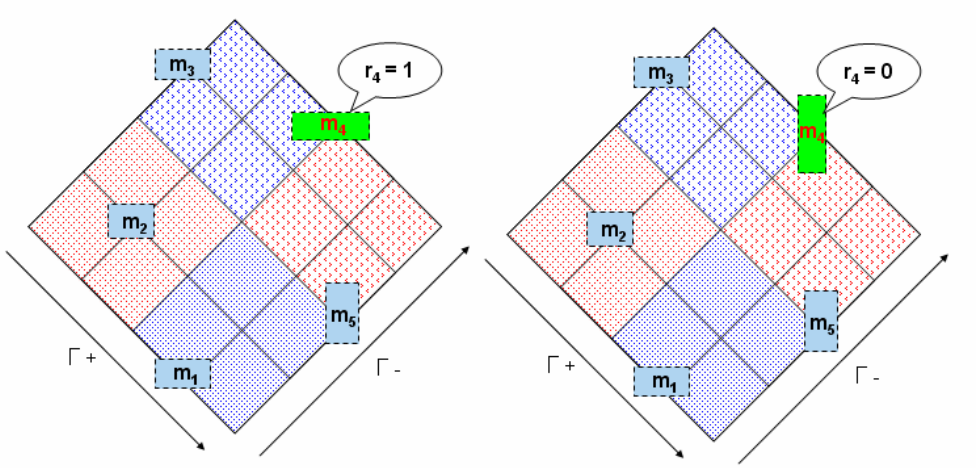

Figure 26: An example of layout before and after "rotation" in focusing search

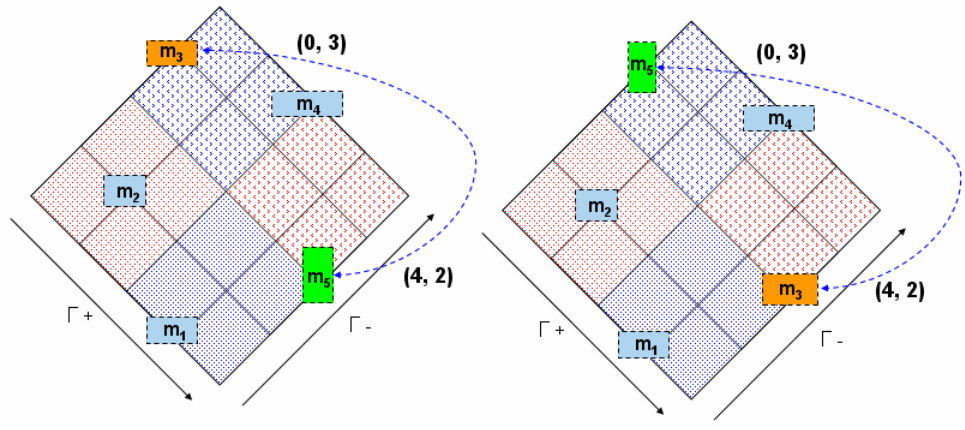

Figure 27: An example of layout before and after "exchange" in focusing search

In detail, the rotation changes the orientation of a module. When a rotation is

applied to module m_i, r_i is changed to 1 - r_i. As an example shown in Figure 26, if a rotation is applied to module m_4, r_4 is changed to 1 –r_4. With respect to 3D packing, r_i is randomly selected from $r_{x\text{-}i}$, $r_{y\text{-}i}$, and $r_{z\text{-}i}$.

The exchange moving method exchanges the order of two modules in Γ^i, where Γ^i corresponds to all sequences, i.e. the sequence pair (Γ^+, Γ^-), the sequence triple (Γ^1, Γ^2, Γ^3) or the sequence quintuple (Γ^1, Γ^2, Γ^3, Γ^4, Γ^5). When an exchange is applied to module m_i and m_j with sequence triple representation, $F^1(m_i)$, $F^2(m_i)$, $F^3(m_i)$, $F^1(m_j)$, $F^2(m_j)$ and $F^3(m_j)$ are changed to $F^1(m_j)$, $F^2(m_j)$, $F^3(m_j)$, $F^1(m_i)$, $F^2(m_i)$and $F^3(m_i)$, respectively. In the case of sequence pair, $F_+(m_i)$, $F\text{-}(m_i)$, $F_+(m_j)$, and $F\text{-}(m_j)$ are changed to $F_+(m_j)$, $F\text{-}(m_j)$, $F_+(m_i)$, and $F\text{-}(m_i)$, respectively. For example, if the modules m_3 and the module m_5 are operated by the exchange in Figure 27, then ($F_+(m_5)$, $F_-(m_5)$) is changed from (4, 2) to (0, 3), and ($F_+(m_3)$, $F_-(m_3)$) is changed from (0, 3) to (4, 2).

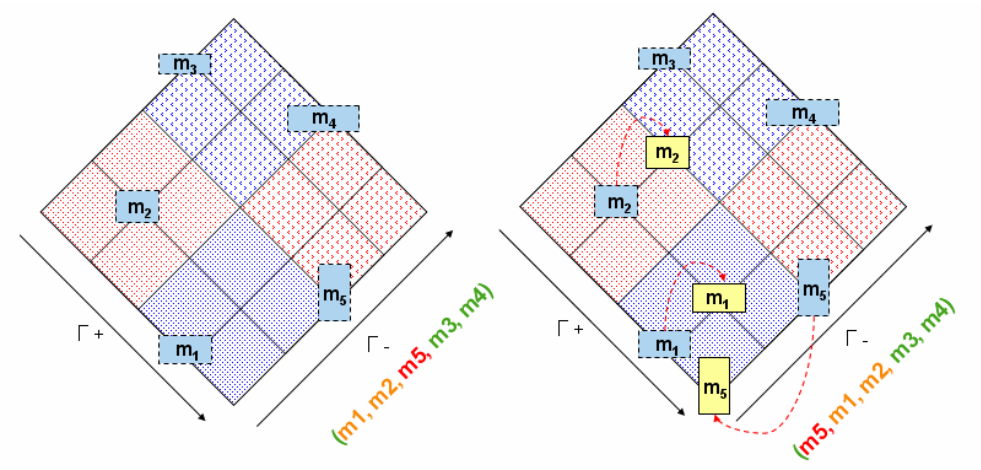

Figure 28: An example of layout before and after “insertion” in focusing search

The insertion changes the order of a module in Γ^i. When a insertion is applied to module m_i in Γ^i, $F^i(m_i)$ is changed to another value, say j, and the orders of modules whose order is between $F^i(m_i)$ and j are shifted accordingly. For example, if the operation is to insert m_5 to $F_-(m_5) = 0$ in Γ^-, the insertion will lead to $F_-(m_1) = F_-(m_1) + 1$ and $F_-(m_2) = F_-(m_2) + 1$, i.e.$\Gamma^-(m_1, m_2, m_5, m_3, m_4)$ is changed to $\Gamma^-(m_5, m_1, m_2, m_3, m_4)$ as shown in Figure 28.

4.2 Paremeter Setting

4.2.1 Scheduling Parameter

In real implementation with a given finite runtime, we are using a fast geometric simulated quenching scheduling ($T_{k+1} = qT_k$, $0<q<1$) with repeated inside loop (p times) to enhance the efficiency of standard SA as the following equation.

$$T_i = T_0 \cdot q^{\lfloor i/p \rfloor}$$

where i is the iterative step, T_i is the variable temperature at the i^{th} step, T_0 is the initial temperature when $i = 0$, p is the inside loop number and q the temperature coefficient near but less than 1, such as $q = 0.99$, $p = 1000$. Generally, a fast annealing of temperature scheduling decreasing exponentially according to $T_i = T_0 \exp(-ci^{1/D})$ is used, where c and D are two constant parameters and T_0 is the initial temperature. This annealing schedule is faster than fast Cauchy annealing, where $T_i = T_0 / i$, and much faster than Boltzmann annealing, where $T_i = T_0 \ln i$.

4.2.2 Guide Parameter

For the guide, there are four different moving methods in this research, so the number of moving methods $l = 4$. The guide is not updated after an iteration loop if no improvement happens during the iteration loop. That is, we only pick up $-\Delta C>0$ to calculate a_k. The initial probability p_0 is same for each moving method. The next selection ratio p_{k+1}, which is defined by $(s_k+p_k)/2$, is adapted according to the current selection ratio p_k and the improvement ratio s_k. To confirm the effectiveness of guide and crossover during the whole search process, the improvement ratio s_k and the selection ratio p_k of each moving method according to temperature (T from T_0 to T_e) is gotten by the experiment using ami98_3D, as shown in Figure 29. Based on the experiment, it is confirmed that the improvement ratio s_k of crossover is normally

bigger than that of any other moving method, when the temperature is large enough. During the initial stage, the crossover is the most efficient moving method, but it is not efficient at the final stage.

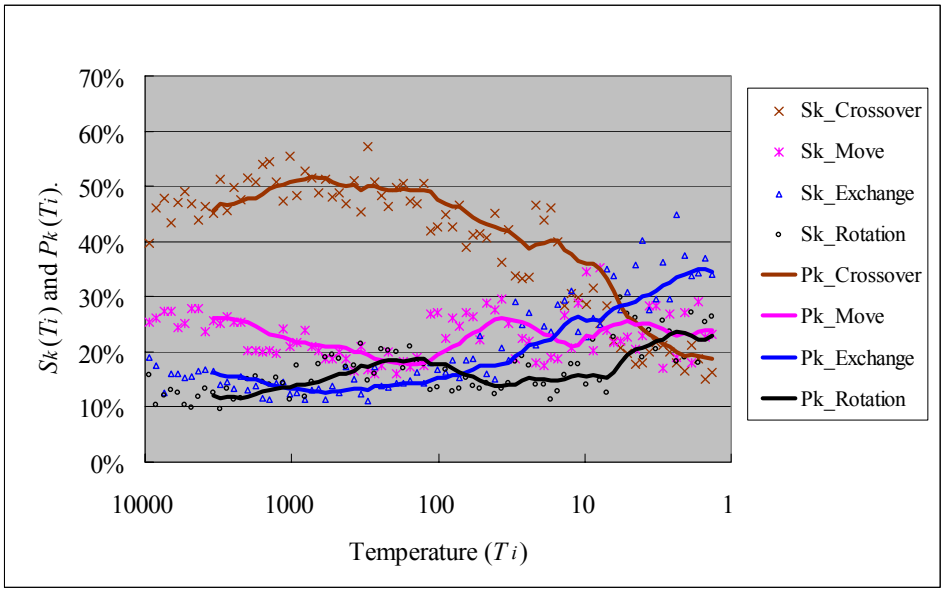

Figure 29: The improvement ratio s_k and the selection ratio p_k according to temperature scheduling

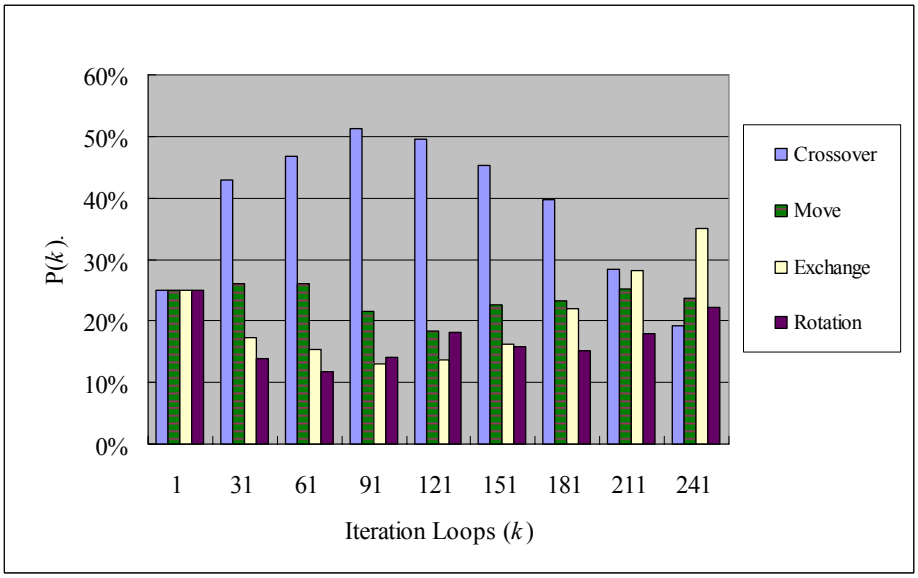

Figure 30: Guide with probabilities $p(k)$ to select each moving method according to the improvement ratio

The selection ratio p_k is a relative value, which is calculated by improvement ratio and the previous selection ratio, and the total probability 100% is maintained.

The initial probability p_0 is set to 25%. The number of trials in each iteration loop is t=300. The selection ratio (p_k) is shown in Figure 30.

For 3D packing optimization, the average results of 50 trials are gotten. All experiments are implemented within 4000s each time using ami98_3D benchmark. We test ASA with and without crossover as shown Table 4. When we are using the crossover operator, the average improvement of 3D packing using SQ representation is near 9.6% with runtime from 20s to 4000s. In short, both the guide and the crossover improve the efficiency of search process.

Table 4: Performance comparison of 3D packing by adaptive simulated annealing with and without crossover

Runtime (s)	ASA	ASA_X	Improvement (%)
20	178.1%	165.7%	12.5%
40	169.0%	148.6%	20.4%
200	151.2%	140.2%	11.0%
400	142.3%	136.5%	5.9%
2000	132.8%	129.3%	3.5%
4000	123.9%	119.6%	4.3%

4.3 Summary

Adaptive simulated annealing with crossover (ASA_X) is proposed to improve the existing simulated annealing based on the combination of moving method and adaptive selection. In the proposed algorithm, a guide with adaptive probabilities is used to select diverse moving methods including a special crossover. Trial experiments are done to set the parameters of the proposed ASA_X, comparing with simulated annealing (SA) and adaptive simulated annealing (ASA).

5. Relay Race Algorithm

This chapter is a new proposed heuristic [60-66] which consists of two sections. The first section introduces the basic parts of relay race algorithm (RRA). The basic idea is similar optimum search with diverse moves. In this section, rough search, focusing search, relay operation and multi-stage problem solver are introduced. The second section is about parameter settings of the proposed RRA.

Relay race algorithm (RRA) is a similar optimum search method with multi-stage improvement which includes three stages: rough search, focusing search and relay. To approach the global optimal solution by exploring more local optimal solutions efficiently, RRA is proposed as the following. Each runner is assigned to find one local optimal solution. A team is made up of a predetermined number of runners. RRA consists of rough search, focusing search and relay. It starts with rough search, which is designed to access one of local optimal solutions as fast as possible. After rough search, focusing search is used to reach the local optimal solution as close as possible. After focusing search, relay is implemented to escape current local optimum. The relay is designed to improve global search ability. After relay, RRA returns to rough search again until a terminal condition. The output is the best value among all searched local optimal solution.

5.1 Similar Optimum Search with Diverse Moves

Relay race algorithm (RRA) is an embodiment of similar optimum search with diverse moves which is introdueced to approach the global optimal solution by exploring local optimal solutions with sophisticated strategies. In the existing methods, a local optimal solution which is similar to an obtained local optimal solution would not be explored. It is expected that a better local optimal solution exists which is similar to an obtained local optimal solution. The efficiency of existing search

methods is limited due to the above reason. RRA consists of rough search, focusing search and relay. Rough search, focusing search, and relay are executed in turn and are repeated in the predetermined number. In rough search and focusing search, a solution is generated by a moving method, and it is accepted if it is better than the current solution. The differences between rough search and focusing search are in moving methods and in terminal condition. In rough search, rough moving methods that modify a current solution relatively large are used. Rough search terminates when the total number of non-improved solutions reaches the predetermined number. Rough search is designed to get over small hills on the solution space and to approach a local optimal solution as fast as possible. In focusing search, focusing moving methods that modify a current solution little are used. Focusing search terminates when no improved solution is found consecutively in the predetermined number. In such cases, the current solution is regarded as a local optimum. The focusing search is designed to reach the local optimal solution as close as possible. Relay is one-time modification and accepted even if a generated solution is far worse than the current solution. Relay is controlled by a global parameter so that the solution generated is far enough from the current local optimum to escape from it without loss of search continuity. The output is the best among all searched local optimal solution.

RRA is essentially different with several existing search algorithms, such as genetic algorithm (GA) and variable neighborhood search (VNS). GA is a directed random search algorithm which is based on the evolution of a population of many individuals. As a population-based algorithm, GA requires a group of initial solutions to form a population, while RRA requires one initial solution. The crossover operator of GA is used to approach the optimum by creating two new individuals as children from two existing individuals as parents, while the relay operator is used to create a new solution which is far enough from the searched local optimal solutions without loss of search continuity. VNS is a search algorithm which is based on the idea of exploiting systematical change of neighborhood within local searches. The search history of VNS is not considered, while RRA includes the relay operator which utilizes search history. Even if VNS uses K neighborhood structures instead of one

neighborhood structure, it is still limited to explore increasingly distant neighbors of the incumbent solution within local searches. In the whole search process, VNS has only one stage for improvement, while RRA includes two dependent stages for improvement. Unlike VNS, the rough search is not used to search a local optimum, but it is designed to get over small hills and to speed up the search process of focusing search by moving to the non-neighborhood state.

5.1.1 Rough Search

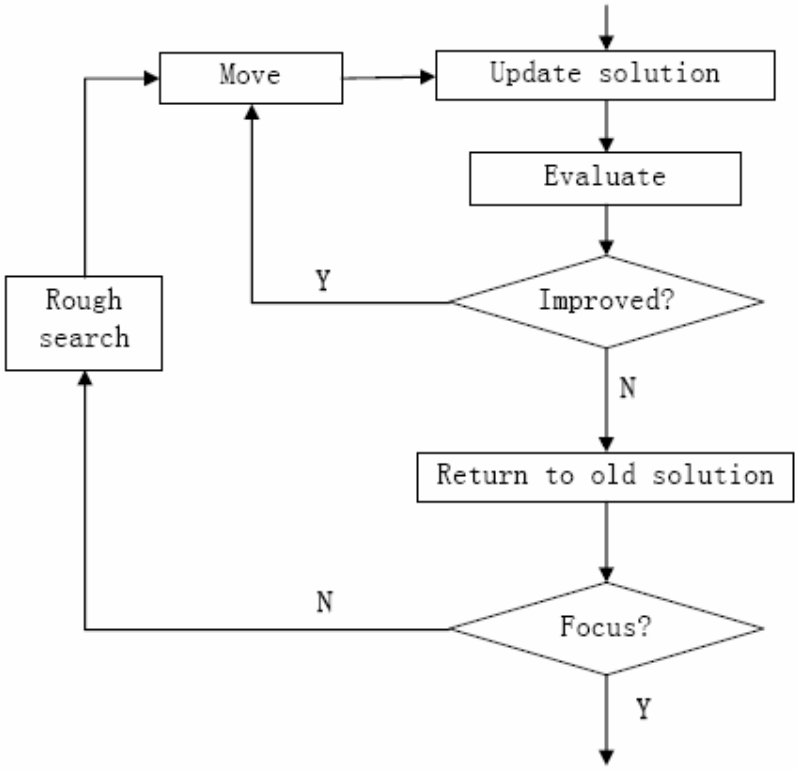

Figure 31: Rough search is the first stage of RRA

Rough search is the first stage of RRA as shown in Figure 31. In rough-search stage, the initial solution is randomly produced and the restarted solution is produced by relay operation as the current solution. A candidate is one of solutions produced by rough moving method operating the current solution. A cost function is used to compare the current solution with the candidate. If the cost is improved, the candidate

replaces the current solution. The terminal condition of rough search is controlled by a predetermined number (N_r). The parameter N_r is the total number of all non-improved trials, which is used to control the maximum scope of a single rough search. The purpose of rough search is to get over small hills on the state-cost distribution.

5.1.2 Focusing Search

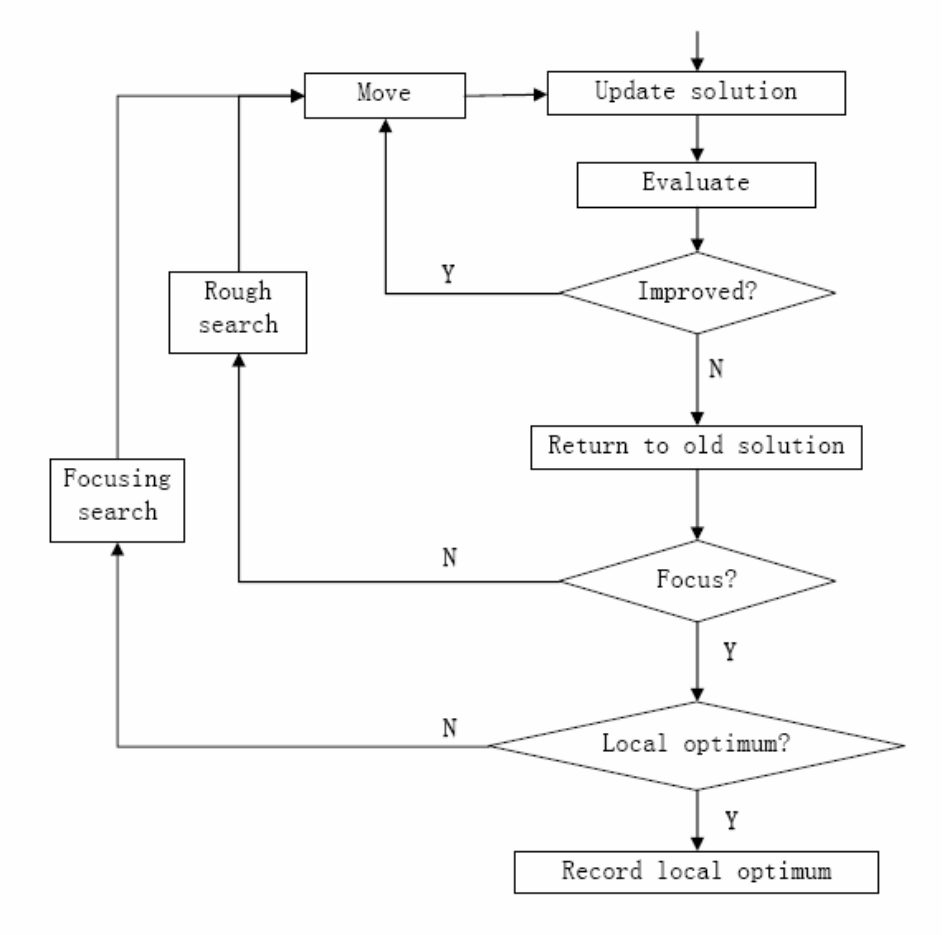

Figure 32: Focusing search is the second stage of RRA.

Focusing search is the second stage of RRA as shown in Figure 32. In focusing-search stage, the initial current solution is given by the last solution of rough search. A candidate is one of solutions produced by focusing moving method operating the current solution. A cost function is used to compare the current solution

with the candidate. If the cost is improved, the candidate replaces the current solution. Focusing search terminates if no better candidate is found when reaching a predetermined number (N_f) of repeated trials. It means N_f times continuous non-improved trials are used to test a local optimum. The purpose of focusing search is to get a local optimum.

5.1.3 Relay Operation

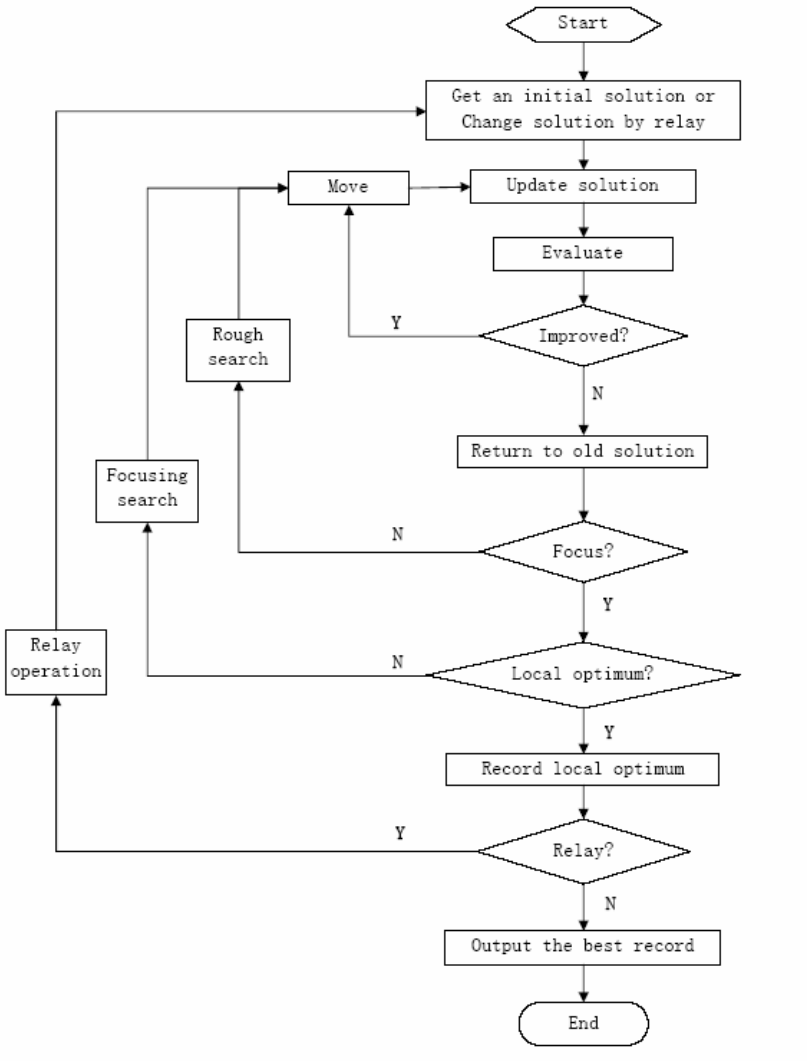

Figure 33: Relay operation is implemented at the end of focusing search to escape a local optimum.

The relay is implemented at the end of focusing search and at the restarting point

of rough search as shown in Figure 33. The state after relay is made up of two parts. One part is randomly produced, though it is also allowed to be given by priori knowledge. Another part is from the latest local optimal solution of relayed runner. When the total number of the relayed runners is up to N_t, it is the terminal condition of algorithm. The purpose of relay is to escape one local optimum and to approach another local optimum. As shown in Figure 34, airplane is an intuitive image of relay operation in the global scope to connect two unconnected points of the solution space using basic moving methods

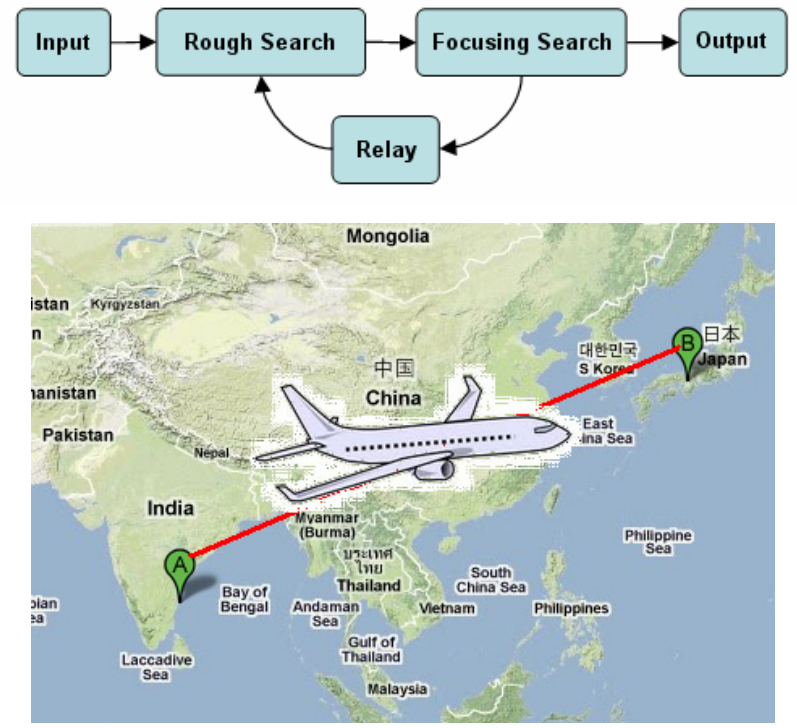

Figure 34: Airplane is an intuitive image of relay operation in the global scope to connect two unconnected points of the solution space using basic moving methods.

5.1.4 Multi-Stage Problem Solver

Generally speaking, multiple stages are normally needed to solve a general problem as follows. (1) Be familiar. What do we know about the problem? That is the key question we should answer carefully to define basic concepts and to get known conditions. In this step, the problem is defined and formulated in an appropriate way.

(2) Be operable. The purpose of this step is to get a solution space no matter feasible or not. Is there a solution space to satisfy all objectives without consideration about constraints? If solution exists, trying to find the solution space with variables and an operable plan. If it does not exists, trying to prove the nonexsitance of solution with an operable plan. In this step, the problem is represented in an appropriate way. (3) Input data. The input is the necessary information including solution space, cost function and moving methods. (4) Be feasible. Is there a feasible solution space to satisfy all constraints? If it exists, trying to find a feasible solution space with variables, moving methods, cost function and a feasible plan. If it does not exists, trying to prove the nonexsitance of feasible solution. (5) Be optimized locally. This step is to find local optimal solutions by an efficient search method such as iterative improvement. (6) Be relayed globally. This step is to find an optimal solution by a global search? All we should do is trying to solve the problem as well as possible, even if it is hard to find the global optimum. (7) Output result. The output is the best of searched local optimal solutions.

Based on the above multiple stages, similar optimum search based problem solver is designed to solve a general problem as follows. (1) Be familiar. The problem should be formulated well to get better understanding of the problem. (2) Be operable. The problem should be represented well as the preparation of problem solving. A good representation should have a relatively small solution space which includes at least one global optimum. (3) Input data. The input of problem solver should be the necessary information for the optimization of the problem. (4) Be feasible. Even if the initial solution is infeasible, it is still possible to search a feasible solution by using rough moving methods as fast as possible. Actually, the solution with a better representation should be feasible. (5) Be optimized locally. The focus-and-relay loop is used to search many local optimal solutions in the global scope. The focusing search is used to search local optimum as close as possible. (6) Be relayed globally. The relay is designed to escape the current local optimum and to keep global search continuity simulteously. (7) Output result. The output is the best of searched local optimal solutions. As we know, a global optimum is the best of all local optimal

solutions, so we have more chance to get the global optimum by searching more local optimal solutions.

In general, all problems are similar due to the following reasons. Any practical problem is originated from the gap between reality and expectation. Accordingly, problem solving is to find a way to eliminate or reduce the gap as well as possible. The hardness of a problem depends on the situation of reality as well as expectations which could be fixed or flexible. If the reality is far from our expectations, the corresponding problem tends to be hard. The expectations are the objectives of a problem which are used to evaluate the correctness or quality. The correctness means to judge “ture” or “false” of a solution, while the quality means to judge “how good” or “how bad” of a solution. The reality is a set of all feasible solutions with all probabilities. The key point to solve the problem is how to search one of the best feasible solutions which satisfy our expectations as close as possible. That is to find the best expectation in the most brutal reality.

There are six starting points of heuristic thinking to solve a general problem similarly and effectively according to our past experience. The first point is the concepts and their definitions. The past experience tells us that some important details of problem might be slipped up due to unclear, equivocal, confusing, inexplicit or inaccurate definitions. A good definition includes the most essential features of a concept. Many problems seem hard only because the solvers did not make the definitions clear. The second point is the conditions of problem. The past experience tells us that it is one of the important things to mine pertinent conditions including hidden conditions because they are the source of reasoning. Only if the conditions are sufficient, it is possible to get the expected conclusion by deduction. The third point is to write down related rules, including laws, formula and theorems. The past experience tells us that it is quite possible to find a way to connect the known conditions and the expected conclusions by tentative deducations using the related rules. The fourth point is to get a better understanding of possible conclusions. The past experience tells us that it is normally helpful to guess a conclusion by intuition or induction. If there are any contradictions between the known conditions and a guessed

conclusion, that means the guess is not correct. The fifth point is the limitations. The past experience tells us that the limitations are a special example, a useful trend or an exception of the problem. And a typical limitation is the infinite version of a finite problem. The sixth point is the assumptions which are based on inspiration, forcast, imagination and heuristic ideas. The past experience tells us that it is not only the most flexible way but also the most universal way to construct extra conditions and any theories.

If all mentioned starting points have been tried but still did not work, what is the next move we can take? The past experience tells us that it is always helpful to ask such a key question: why it is hard to find a good start? It is worthy to thinking it over. Is the problem too complex? Is the field of problem totally unknown? Or is the description of problem too hard to understand. After we figure the key question out, we will get a thinking direction to figure the problem out. If the problem is too complex, let us make it simple or try a simpler version. If the field of problem is totally unknown, let us transform it to familiar fields. If the description of problem is too hard to understand, let us make the description easier to understand. Anyway, we get a right direction to solve the problem. That is the power of cause and effect which is one of heuristic thinking chains.

Based on the above discussion, three chains of heuristic thinking are summarized as follows. The first chain is the cause of cause. The direction of cause is the past, while the direction of effect is the future. In fact, causes and effects are the foundation of any rational thinking including problem solving. The second chain is the abstract of abstract. The abstract is related with our brains. We always construct some abstract models to get better understanding of the real world, which is the final object of abstract. The third chain is the inversion of inversion. A state after the inversion of inversion is not completely the original state but an updated state which is supposed to be a simlar but improved state. For example, the inversion of one cause is an effect, but the inversion of the effect might be another cause. The inversion of one type of abstract is embodiment, but the inversion of embodiment might be another type of abstract. Based on our knowledge, any stable things in the real world are swinging

between something and its inversion. It is a basic feature of a thing to be stable; otherwise it will disappear or be changed to other stable things sooner or later. In fact, the inversion of inversion is the foundation of similarity which is the main focus of this research. The chains of heuristic thinking tell us on how to think about problem solving more effectively as well as the real world itself.

5.2 Parameter Setting

For the parameter setting of RRA, the best empirical values of parameters N_f, N_r, and R_e are investigated by trial experiments of placement using ami49 benchmark. Although the empirical values of parameters are no longer valid for different problems with different sizes or different objectives, the method below to get the values is still applicable.

The parameter R_e is the percentage of the new part of sequence within two sequences. For example, if R_e = 1/3, then 1/3 part of each sequence is randomly produced to get a new sequence. The rotation and the exchange are defined as two basic moving methods. In the case of two sequences, the ΔS_{max} is equal to $4n$, because it takes at least $2n$ times repeated exchange operations plus at least $2n$ times repeated rotation operations from a state to any other states, where n is the total number of modules. If R_e is near zero, ΔC and $dis(S_b, S_a)$ are near zero too, where S_a is a state before relay and S_b is a state after relay. If R_e = 1, ΔC and $dis(S_b, S_a)$ are totally random. Although Re could be any value between 0 and 1, Re = 1/q is used to implement trial experiments this time, where q is a positive integer. The selection of the random part for Γ+, Γ- and rotation is set to be same. For example, if q = 4 (i.e. Re = 0.25) for Γ+, one of four following parts Γ+ [0, n/4-1], Γ+[n/4, n/2-1], Γ+[n/2, 3n/4-1] and Γ+[3n/4, n-1] is randomly selected, where n is the number of modules. If Γ+[n/4, n/2-1] is selected, then Γ-[n/4, n/2-1] and the rotation of [n/4, n/2-1] are also selected.

As shown in the Figure 35, train, bus and walk are intuitive images of multi-stage improvement in the local scope using diverse moving methods. Three

focusing moving methods including rotation, exchange and insertion are defined as follows. Firstly, the rotation changes the orientation of a module. When a rotation is applied to module m_i, the orientation r_i is changed to 1 - r_i. Secondly, the exchange moving method exchanges the order of two modules in all sequences, e.g. in sequence pair (Γ^1, Γ^2), $F^1(m_i)$, $F^2(m_i)$, $F^1(m_j)$, and $F^2(m_j)$ are changed to $F^1(m_j)$, $F^2(m_j)$, $F^1(m_i)$, and $F^2(m_i)$, respectively. Thirdly, the insertion changes the order of a module in one sequence Γ^i. When a insertion is applied to module m in Γ^i, $F^i(m)$ is changed to another value, say j, and the orders of modules whose order is between $F^i(m)$ and j are shifted accordingly.

Figure 35: Train, bus and walk are intuitive images of multi-stage improvement in the local scope by using diverse moving methods

To improve the search efficiency of RRA, three rough moving methods, named as group rotation, group exchange and group insertion, are designed as follows. The group rotation is repeated rotations of randomly selected modules with a given number. The group exchange is repeated exchanges of randomly selected pairs of modules with a given number. The group insertion is repeated insertions of randomly selected pairs of modules in a selected sequence with a given number. The given number is at least ten in this research to satisfy the recommended condition that ΔM_r is much larger than ΔM_f.

5.2.1 Focusing Parameter

In order to get the best value of N_f, the trial experiments of focusing search stage is executed without rough search and relay. All initial solutions are randomly generated. As shown in Table 5, Table 6 and Table 7, the number of trails in focusing search stage increases as N_f increases. The cost of placement obtained by focusing search stage is decreasing continuously when N_f increases from 0 to 1000. However, the cost is almost same when N_f is larger than 1000. This shows that a local optimal solution is obtained when N_f is set larger than 1000. So we set N_f=1000 for ami49 benchmark.

Table 5: Trial experiment of focus parameter setting in the best case (20 trials)

N_f	Number of Trials	Number of Acceptance	Number of Rejection	Runtime (s)	Cost	
					Start	End
1	11	10	1	0.01	2.73	2.65
5	199	148	51	0.24	2.73	2.59
10	1721	1073	649	1.98	2.73	2.41
50	6938	2917	4021	8.29	2.73	2.27
100	14414	4768	9646	16.34	2.73	2.02
500	27590	7727	19863	31.67	2.73	1.51
1000	53928	10381	43547	64.15	2.73	1.37
5000	96655	10535	86120	113.83	2.73	1.37
10000	143473	10685	132788	171.26	2.73	1.37

Table 6: Trial experiment of focus parameter setting in the average case (20 trials)

N_f	Number of Trials	Number of Acceptance	Number of Rejection	Runtime (s)	Cost	
					Start	End
1	6	5	1	0.01	2.73	2.69
5	221	163	58	0.26	2.73	2.64
10	1885	1169	716	2.19	2.73	2.44
50	7561	3204	4357	8.79	2.73	2.33
100	15534	5203	10331	18.06	2.73	2.07
500	29539	8778	20761	34.35	2.73	1.56
1000	58216	11385	46831	67.70	2.73	1.41
5000	106131	11772	94359	123.42	2.73	1.43
10000	156782	11998	144784	182.32	2.73	1.44

Table 7: Trial experiment of focus parameter setting in the worst case (20 trials)

N_f	Number of Trials	Number of Acceptance	Number of Rejection	Runtime (s)	Cost	
					Start	End
1	2	1	1	0.01	2.73	2.72
5	235	172	63	0.28	2.73	2.70
10	2039	1226	813	2.35	2.73	2.50
50	8086	3481	4605	9.64	2.73	2.38
100	17068	5681	11387	19.01	2.73	2.11
500	31367	9270	22097	37.43	2.73	1.57
1000	63041	12387	50654	72.67	2.73	1.44
5000	115538	12400	103138	135.56	2.73	1.48
10000	168246	12874	155372	200.23	2.73	1.51

5.2.2 Rough Parameter

To get the best value of N_r, the trial experiments of rough search stage is executed without relay, but focusing search stage where $N_f = 1000$ follows. The same initial solution randomly generated is used. As shown in Table 8, Table 9, Table 10, Table 11, Table 12 and Table 13, the cost of solution at the end of rough search stage decreases as N_r increases, but the cost of a solution at the end of focusing search stage

is almost same for all cases. Also, as N_r increases, the number of trials in focusing search stage decreases. This shows the usefulness of rough search stage in order to approach local optimal solution as fast as possible. However, the number of trials in rough search stage increases. The total number of trials is decreasing when N_r increases from 0 to 100. When N_r is larger than 100, the total number of trials starts to increase. So we set N_r=100 for ami49 benchmark.

Table 8: Trial experiment of rough parameter setting (N_f=1000) in the best case (20 trials)

N_r	Number of Trials			Runtime (s)	Cost		
	Rough	Focus	Total		Start	End of Rough	End of Focus
1	12	55355	55367	64.32	2.73	2.47	1.37
5	20	52441	52461	60.37	2.73	2.34	1.42
10	63	44700	44763	52.00	2.73	2.23	1.36
50	486	22164	22650	27.97	2.73	1.77	1.38
100	2323	13144	15467	24.54	2.73	1.69	1.31
500	5343	11437	16780	35.53	2.73	1.63	1.37
1000	11828	10461	22289	58.39	2.73	1.48	1.32
5000	23571	9330	32901	104.74	2.73	1.46	1.39
10000	43293	6044	49337	179.49	2.73	1.47	1.37

Table 9: Trial experiment of rough parameter setting (N_f=1000) in the best case (20 trials)

N_r	Number of Trials			Number of Acceptance			Number of Rejection		
	Rough	Focus	Total	Rough	Focus	Total	Rough	Focus	Total
1	12	55355	55367	11	10229	10240	1	45126	45127
5	20	52441	52461	16	10295	10311	5	42147	42152
10	63	44700	44763	54	9894	9948	9	34806	34815
50	486	22164	22650	441	7532	7973	45	14632	14677
100	2323	13144	15467	2229	5777	8006	94	7367	7461
500	5343	11437	16780	4870	3882	8752	473	7555	8028
1000	11828	10461	22289	10912	1918	12830	915	8543	9458
5000	23571	9330	32901	19048	1038	20086	4523	8293	12816
10000	43293	6044	49337	34128	480	34608	9165	5565	14730

Table 10: Trial experiment of rough parameter setting (N_f=1000) in the average case (20 trials)

N_r	Number of Trials			Runtime (s)	Cost		
	Rough	Focus	Total		Start	End of Rough	End of Focus
1	6	60462	60468	67.89	2.73	2.68	1.44
5	16	58122	58138	66.12	2.73	2.51	1.48
10	60	48474	48534	56.32	2.73	2.37	1.41
50	537	23915	24452	29.73	2.73	1.96	1.45
100	2569	14925	17494	27.05	2.73	1.81	1.38
500	6066	13101	19167	37.72	2.73	1.76	1.43
1000	12625	11554	24179	62.51	2.73	1.62	1.39
5000	26733	10755	37488	112.43	2.73	1.57	1.47
10000	47227	6919	54146	190.25	2.73	1.51	1.42

Table 11: Trial experiment of rough parameter setting (N_f=1000) in the average case (20 trials)

N_r	Number of Trials			Number of Acceptance			Number of Rejection		
	Rough	Focus	Total	Rough	Focus	Total	Rough	Focus	Total
1	6	60462	60468	5	11616	11621	1	48846	48847
5	16	58122	58138	11	11647	11658	5	46475	46481
10	60	48474	48534	50	11066	11116	10	37409	37419
50	537	23915	24452	486	8139	8625	51	15775	15826
100	2569	14925	17494	2466	6457	8923	104	8467	8571
500	6066	13101	19167	5564	4264	9828	502	8836	9338
1000	12625	11554	24179	11580	2138	13718	1044	9417	10461
5000	26733	10755	37488	21544	1166	22710	5189	9589	14778
10000	47227	6919	54146	37143	537	37680	10085	6382	16467

Table 12: Trial experiment of rough parameter setting (N_f=1000) in the worst case (20 trials)

N_r	Number of Trials			Runtime (s)	Cost		
	Rough	Focus	Total		Start	End of Rough	End of Focus
1	2	63495	63497	72.65	2.73	2.69	1.47
5	12	62112	62124	72.65	2.73	2.56	1.50
10	62	52221	52283	59.48	2.73	2.43	1.44
50	570	26097	26667	31.48	2.73	1.99	1.47
100	2772	15587	18359	29.11	2.73	1.86	1.39
500	6341	13700	20041	40.72	2.73	1.81	1.44
1000	13554	12375	25929	66.83	2.73	1.67	1.41
5000	27507	11119	38626	123.36	2.73	1.61	1.49
10000	49193	6999	56192	204.83	2.73	1.58	1.45

Table 13: Trial experiment of rough parameter setting (N_f=1000) in the worst case (20 trials)

N_r	Number of Trials			Number of Acceptance			Number of Rejection		
	Rough	Focus	Total	Rough	Focus	Total	Rough	Focus	Total
1	2	63495	63497	1	12256	12257	1	51239	51240
5	12	62112	62124	6	11882	11888	5	50230	50235
10	62	52221	52283	51	11724	11775	11	40497	40508
50	570	26097	26667	515	8752	9267	55	17345	17400
100	2772	15587	18359	2662	6969	9631	110	8618	8728
500	6341	13700	20041	5798	4355	10153	543	9345	9888
1000	13554	12375	25929	12475	2173	14648	1079	10202	11281
5000	27507	11119	38626	22247	1225	23472	5260	9893	15153
10000	49193	6999	56192	38492	556	39048	10701	6443	17144

5.2.3 Relay Parameter

To get the best value of R_e, the trial experiments of RRA with N_f=1000, N_r=100, and N_t=10 are executed. All initial solutions are randomly generated. In Table 3, the results of R_e = 1.00 are shown. Each row corresponds to a runner in RRA. A solution generated by relay is random, and no search history is utilized. The number of trials and the cost of a solution at each stage vary among runners, but the deviation is not so large. In Table 14, Table 15, Table 16, Table 17, Table 18, Table 19, Table 20, Table 21, Table 22, Table 23, Table 24 and Table 25, the best, average and worst results of R_e = 1.00, 0.20, 0.10 and 0.05 are shown respectively. Each row corresponds to a runner in RRA as well. A solution generated by relay is generated by the solution obtained by the previous runner. In each relay, each sequence of SP of the previous runner is divided into 1, 5, 10, and 20 parts respectively, and one randomly selected part is randomly generated. As relay is repeated, the number of trials, the start cost, and the end cost of each runner decrease. This shows that the RRA explores near local optimal solutions efficiently by utilizing search history. Several trial experiments were executed by changing R_e, and we select R_e = 1/10 for ami49 benchmark that achieves

the best final cost by 10 runners.

Table 14: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=1) in the best case

N_t	Number of trials			Runtime (s)	Cost (Re=1)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	1985	13340	15325	23.22	2.73	1.77	1.62
2	2505	13875	16380	26.78	2.73	1.69	1.50
3	2198	15915	18113	28.16	2.73	1.81	1.44
4	1889	13065	14954	22.84	2.73	1.85	1.53
5	2204	16323	18527	27.44	2.73	1.91	1.36
6	2025	11045	13070	20.81	2.73	1.83	1.57
7	2567	15760	18327	28.96	2.73	1.54	1.38
8	2279	16831	19110	28.60	2.73	1.73	1.33
9	2423	11014	13437	22.61	2.73	1.86	1.63
10	2526	16951	19477	29.14	2.73	1.64	1.32

Table 15: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.2) in the best case

N_t	Number of trials			Runtime (s)	Cost (Re=0.2)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2148	13914	16062	24.00	2.73	1.75	1.57
2	1171	10907	12079	17.42	2.08	1.54	1.43
3	880	10269	11149	15.36	1.93	1.58	1.37
4	552	9462	10014	12.92	1.76	1.50	1.35
5	621	9471	10092	13.48	1.90	1.51	1.30
6	468	8675	9143	11.83	1.80	1.45	1.33
7	537	8778	9315	11.90	1.87	1.39	1.26
8	464	8219	8683	11.24	1.70	1.42	1.19
9	435	7263	7698	10.25	1.66	1.33	1.23
10	441	7897	8338	10.67	1.59	1.36	1.13

Table 16: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.1) in the best case

N_t	Number of trials			Runtime (s)	Cost (Re=0.1)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2192	14036	16228	24.69	2.73	1.79	1.62
2	947	10363	11311	16.20	1.91	1.50	1.44
3	649	9115	9764	13.08	1.77	1.51	1.37
4	340	8656	8996	11.38	1.54	1.44	1.31
5	354	8118	8472	10.98	1.67	1.44	1.27
6	227	8178	8405	10.15	1.55	1.35	1.25
7	196	7605	7801	9.68	1.64	1.35	1.22
8	168	6760	6928	8.52	1.48	1.32	1.18
9	134	6512	6646	8.19	1.37	1.20	1.11
10	118	6317	6435	7.91	1.40	1.27	1.10

Table 17: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.05) in the best case

N_t	Number of trials			Runtime (s)	Cost (Re=0.05)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2254	12483	14737	24.08	2.73	1.67	1.53
2	861	9364	10225	14.32	1.61	1.49	1.40
3	536	8293	8829	11.77	1.55	1.45	1.37
4	346	7629	7975	10.17	1.55	1.42	1.39
5	284	6603	6887	9.08	1.56	1.42	1.32
6	177	6057	6234	7.58	1.56	1.41	1.30
7	222	6414	6636	8.36	1.53	1.39	1.29
8	108	5732	5840	6.96	1.52	1.38	1.26
9	116	5547	5663	6.59	1.50	1.40	1.30
10	195	4736	4931	6.36	1.48	1.36	1.25

Table 18: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=1) in the average case

N_t	Number of trials			Runtime (s)	Cost (Re=1)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2131	14194	16325	24.90	2.58	1.85	1.67
2	2772	15083	17855	28.46	2.84	1.76	1.54
3	2413	17643	20056	30.02	2.55	1.89	1.50
4	2097	14081	16178	24.63	2.68	1.91	1.59
5	2368	17518	19886	29.70	2.80	1.96	1.42
6	2133	12087	14220	22.46	2.66	1.92	1.65
7	2819	16841	19660	30.69	2.88	1.62	1.42
8	2524	17918	20442	30.78	2.61	1.80	1.37
9	2604	12178	14782	24.42	2.87	1.92	1.68
10	2731	18056	20787	31.75	2.53	1.69	1.38

Table 19: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.2) in the average case

N_t	Number of trials			Runtime (s)	Cost (Re=0.2)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2347	14858	17205	26.52	2.73	1.80	1.63
2	1269	11771	13040	18.69	2.16	1.61	1.49
3	948	10811	11759	16.30	1.98	1.65	1.44
4	608	10025	10633	14.05	1.84	1.57	1.40
5	655	10119	10774	14.35	1.97	1.59	1.34
6	507	9220	9727	12.72	1.84	1.52	1.37
7	572	9280	9852	13.05	1.97	1.44	1.30
8	504	8803	9307	12.22	1.78	1.46	1.25
9	482	7910	8392	11.09	1.73	1.39	1.27
10	487	8351	8838	11.63	1.67	1.40	1.18

Table 20: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.1) in the average case

N_t	Number of trials			Runtime (s)	Cost (Re=0.1)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2381	14964	17345	26.78	2.73	1.79	1.62
2	1029	11243	12072	17.13	1.97	1.57	1.48
3	714	9721	10235	14.12	1.83	1.58	1.43
4	371	9378	9549	12.37	1.61	1.48	1.35
5	382	8939	9321	11.90	1.75	1.49	1.32
6	248	8763	8911	11.17	1.62	1.41	1.3
7	214	8074	8288	10.23	1.72	1.39	1.27
8	182	7349	7531	9.26	1.55	1.37	1.22
9	143	7229	7372	8.97	1.42	1.25	1.16
10	129	6803	6932	8.42	1.44	1.32	1.13

Table 21: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.05) in the average case

N_t	Number of trials			Runtime (s)	Cost (Re=0.05)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2586	14228	16814	25.89	2.73	1.73	1.58
2	972	9986	10958	15.15	1.68	1.56	1.45
3	622	9205	9827	12.86	1.61	1.50	1.43
4	382	8443	8825	11.18	1.62	1.49	1.44
5	311	7593	7904	9.62	1.60	1.47	1.38
6	197	6526	6723	8.29	1.61	1.46	1.35
7	241	7249	7490	9.03	1.59	1.44	1.33
8	125	6296	6421	7.66	1.58	1.43	1.30
9	133	5972	6105	7.29	1.55	1.47	1.34
10	220	5275	5495	6.86	1.52	1.43	1.29

Table 22: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=1) in the worst case

N_t	Number of trials			Runtime (s)	Cost (Re=1)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2324	15502	17826	26.59	2.73	1.91	1.73
2	3039	16371	19410	30.88	2.73	1.80	1.59
3	2584	19158	21742	32.33	2.73	1.94	1.55
4	2269	15166	17435	26.87	2.73	1.98	1.67
5	2539	18914	21453	31.86	2.73	2.03	1.47
6	2291	13223	15514	23.98	2.73	1.98	1.70
7	3020	18225	21245	32.93	2.73	1.69	1.47
8	2677	19669	22346	32.60	2.73	1.88	1.41
9	2763	13152	15915	26.01	2.73	1.98	1.75
10	2956	19083	22038	34.21	2.73	1.73	1.45

Table 23: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.2) in the worst case

N_t	Number of trials			Runtime (s)	Cost (Re=0.2)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2484	16100	18584	28.46	2.73	1.85	1.70
2	1388	12573	13961	19.98	2.26	1.66	1.57
3	1031	11538	12569	17.63	2.07	1.71	1.49
4	655	10735	11390	15.04	1.91	1.64	1.46
5	691	11018	11709	15.33	2.03	1.66	1.40
6	551	9867	10418	13.97	1.91	1.58	1.43
7	605	10113	10718	13.92	2.05	1.48	1.36
8	531	9335	9866	13.34	1.86	1.51	1.29
9	519	8307	8826	11.78	1.79	1.45	1.31
10	528	8938	9466	12.71	1.75	1.45	1.22

Table 24: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.1) in the worst case

N_t	Number of trials			Runtime (s)	Cost (Re=0.1)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2552	15740	18292	28.90	2.73	1.79	1.62
2	1125	12058	13183	18.54	2.07	1.64	1.53
3	784	10215	10999	15.06	1.88	1.65	1.47
4	402	10072	10474	13.49	1.69	1.53	1.39
5	402	9571	9973	12.81	1.82	1.54	1.38
6	272	9427	9699	12.27	1.67	1.45	1.36
7	231	8490	8721	11.18	1.77	1.44	1.33
8	195	8017	8212	9.79	1.61	1.41	1.26
9	157	7785	7942	9.53	1.47	1.30	1.21
10	138	7380	7518	9.06	1.48	1.36	1.17

Table 25: Trial experiment of relay parameter setting (N_r=100, N_f=1000, Re=0.05) in the worst case

N_t	Number of trials			Runtime (s)	Cost (Re=0.05)		
	Rough	Focus	Total		Start of Rough	End of Rough	End of Focus
1	2725	14771	17496	27.54	2.73	1.77	1.65
2	1016	10522	11538	16.38	1.73	1.64	1.49
3	641	9663	10304	13.98	1.66	1.57	1.47
4	405	9073	9478	11.88	1.70	1.54	1.50
5	323	7881	8204	10.46	1.67	1.54	1.43
6	206	6967	7173	8.94	1.66	1.53	1.42
7	251	7595	7846	9.79	1.66	1.48	1.39
8	129	6695	6824	8.23	1.64	1.50	1.36
9	137	6314	6451	7.88	1.60	1.54	1.41
10	229	5635	5864	7.37	1.57	1.48	1.36

5.3 Summary

A novel high-performance heuristic called relay-race algorithm (RRA) is proposed to approach global optimum efficiently by introducing rough moving methods and relay with sophisticated strategy. The proposed RRA consists of three stages: rough search, focusing search and relay. The rough search is designed to get over small hills on the solution space and to approach a local optimal solution as fast

as possible. The focusing search is designed to reach the local optimal solution as close as possible. The relay is designed to escape the local optimal solution in only one step without loss of search continuity.

6. Applications to Physical Design Optimization

This chapter is the applications of improved search methods to physical design optimization [31-59] which consists of three sections. The first section is the formulation and representation of packing and placement. In the second section, the experiment of packing optimization is introduced to comfirm the efficiency of ASA_X. The last section is the experiment of placement optimization which is introduced to comfirm the efficiency of RRA.

6.1 Packing and Placement Optimizaiton

6.1.1 Problem Formulation

2D packing is a special case of 3D packing, so let us start the formulation of 3D case. 3D rectangular packing problem is formulated as follows. Let $M = \{m_1, m_2, ..., m_n\}$ denote the modules or blocks to be placed, where n is the number of modules. Each m_i, where $1 \le i \le n$, has height h_i, length l_i and width w_i. Let $(x_i, y_i, z_i, r_{xy\text{-}i}, r_{yz\text{-}i}, r_{zx\text{-}i})$ of module m_i be the location and rotation on 3D orthogonal coordinate system, where (x_i, y_i, z_i) means the coordinates of the bottom-south-west corner of module m_i, and $(r_{xy\text{-}i}, r_{yz\text{-}i}, r_{zx\text{-}i})$ represents the rotation (0, 1) of m_i on xy-, yz- and zx- plane. If $r_{xy\text{-}i} = 1$, the height is the vertical length and the width is the horizontal length on xy- plane. If $r_{xy\text{-}i} = 0$, the height will be the horizontal length and the width will be the vertical length, which is rotated by 90 degree on xy- plane. In short, the input is a set of modules $M = \{m_1, m_2, ..., m_n\}$ with height, length and width $\{(h_1, l_1, w_1), (h_2, l_2, w_2), ... , (h_n, l_n, w_n)\}$. The constraint is no overlap between m_i and m_j, where $i \ne j$. The output is a set of location and rotation of modules $\{(x_1, y_1, z_1, r_{xy\text{-}1}, r_{yz\text{-}1}, r_{zx\text{-}1}), (x_2, y_2, z_2, r_{xy\text{-}2}, r_{yz\text{-}2}, r_{zx\text{-}2}), ..., (x_n, y_n, z_n, r_{xy\text{-}n}, r_{yz\ n}, r_{zx\text{-}n})\}$. The objective is to minimize the volume of bounding box.

As a special case, 2D rectangular packing is simply formulated as follows. The input is a module set $M = \{m_1, m_2, \ldots, m_n\}$ with the height and width $\{(h_1, w_1), (h_2, w_2), \ldots, (h_n, w_n)\}$. The constraint is no overlap between m_i and m_j, where $i \neq j$. The output is the set of location and rotation for each module $\{(x_1, y_1, r_1), (x_2, y_2, r_2), \ldots, (x_n, y_n, r_n)\}$ on the plane. The objective is to minimize the bounding area.

6.1.2 Problem Representation

In general, let $A+B$ be the sequence which is the concatenation of A and B, and $A-B$ be the sequence obtained from A by removing all the elements in B, where A and B are sequences. Let us denote $A[i, j]$, where $i<j$, as the sequence $(A[i], A[i+1], \ldots, A[j])$, where $A = (A[0], A[1], \ldots, A[n-1])$. Let $\Gamma^i = (\Gamma^i[0], \Gamma^i[1], \ldots, \Gamma^i[n-1])$ be a sequence$(1 \leq i \leq v)$, where v is the number of sequences. Let $F^i(m_j)$ be the order of m_j in sequence Γ^i. For example, if $\Gamma^i[l]$ is m_j, then $F^i(m_j) = l$. So the order of m_j can be represented by $(F^1(m_j), F^2(m_j), \ldots, F^v(m_j))$.

The original 2D/3D packing problem is with infinite solution space. The coding and decoding method [42][46] is needed to connect the problem and its representation. The solution space of a good representation should be finite. The solutions after a good representation should be feasible and be better to include at least one optimal solution. Sequence pair (SP) (Γ^1, Γ^2) represents a general 2D rectangular topology. Two sequences generate a finite solution space which includes at least one optimal solution of 2D topology for area optimization by decoding. It is regarded as a set of the relations of relative location between modules, i.e. “North-South” and “West-East” (NS- and WE-) relations. Let $(m_i \text{ N } m_j)$ and $(m_i \text{ W } m_j)$ denote NS- and WE-relations. SP defines $(m_i \text{ W } m_j)$ when

$$F^1(m_i) < F^1(m_j) \text{ and } F^2(m_i) < F^2(m_j).$$

It defines $(m_i \text{ N } m_j)$ when

$$F^1(m_i) < F^1(m_j) \text{ and } F^2(m_i) > F^2(m_j).$$

The rule of symmetry to be followed is that $(m_i \text{ N } m_j)$ is the same relation as $(m_j \text{ S } m_i)$. That is to say, the topology should be reversely decoded if the order of labeling is

reversed. For a given packing with n modules, the solution space is $(n!)^2$. If the rotation of the module is not fixed, then the solution space will increase to $(n!)^2 2^n$.

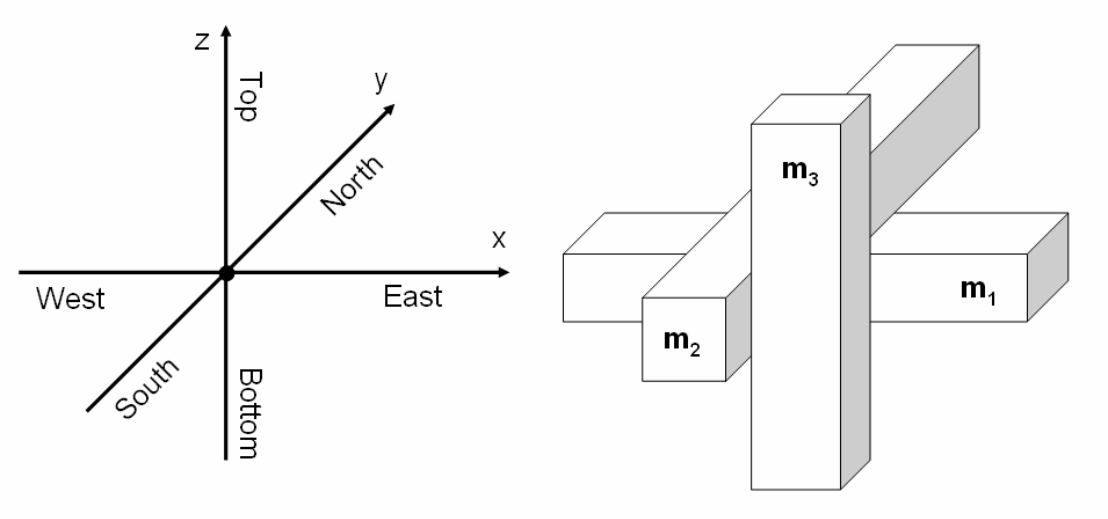

Figure 36: Orthogonal coordinate system and 3D packing topology represented by $\Gamma^1(m_2,m_1,m_3)$, $\Gamma^2(m_1,m_3,m_2)$ and $\Gamma^3(m_1,m_2,m_3)$ according to relative location.

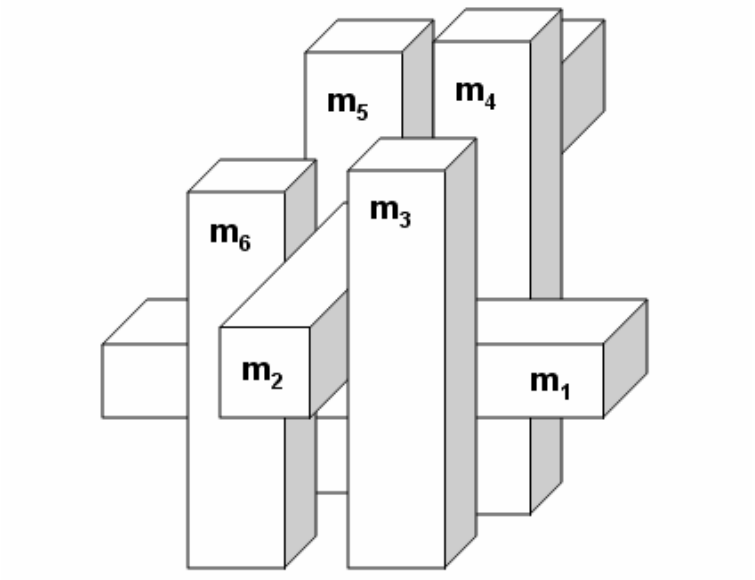

Figure 37: A 3D packing topology that cannot be represented by sequence triple.

SP representation for 2D rectangular packing is extended to 3D packing as sequence triple (ST) representation and sequence quintuple (SQ) representation. The ST and SQ representations define the orthogonal coordinate system (x, y, z) for 3D

rectangular topology, which can be regarded as a set of the relations of relative location between boxes, i.e. “top”, “bottom”, “north”, “south”, “east” and “west” (TB-, NS- and WE-) relations, as shown in Figure 36. For example, box m_2 is on the west of box m_3, since the x-coordinate of any part of box m_2 is always smaller than or equal to that of any part of box m_3. Similarly box m_1 is on the north of box m_3, and box m_2 is on the top of box m_1. ST consists of three sequences Γ^1, Γ^2 and Γ^3. The coding and the decoding are based on TB-, NS- and WE- relation corresponding to the order of modules in ST. For a given packing with n modules, the solution space is $(n!)^3$. If the rotation of the module is not fixed, then the solution space will increase to $(n!)^3 6^n$.

However, ST representation does not cover all kinds of topology of 3D packing. As shown in Figure 37, (m_4 N m_1) and (m_1 N m_6) lead to (m_4 N m_6), while (m_6 W m_2) and (m_2 W m_4) lead to (m_4 W m_6). The pair (m_4, m_6) is conflicting with the rule of uniqueness, i.e. each pair of modules should be assigned with a unique topology. That means the packing can not be represented by sequence triple. As a result, the topology with the minimum volume might not be covered by sequence triple. Three or even four sequences are not enough to represent a general 3D packing topology. For example, the packing in Figure 37 cannot be represented by ST representation. SQ generates a finite solution space which includes at least one optimal solution of 3D packing for volume minimization by decoding. The packing in Figure 37 is represented by SQ representation, where Γ^1=(m_1, m_5, m_6, m_2, m_4, m_3), Γ^2=(m_6, m_5, m_2, m_3, m_4, m_1), Γ^3=(m_2, m_4, m_5, m_1, m_3, m_6), Γ^4=(m_5, m_4, m_1, m_6, m_3, m_2) and Γ^5=(m_6, m_5, m_4, m_3, m_2, m_1). The coding and the decoding are based on TB-, NS- and WE- relation corresponding to the order of modules in SQ. For a given packing with n modules, the solution space is $(n!)^5$. If the rotation of the module is not fixed, then the solution space will increase to $(n!)^5 6^n$.

In case of sequence triple, it consists of three independent sequences Γ^i, where $1 \le i \le 3$. The coding and the decoding are based on TB-, NS- and WE- relation corresponding to the order of modules. Sequence triple defines (m_i W m_j) when

$$F^2(m_i) > F^2(m_j) \text{ and } F^3(m_i) < F^3(m_j).$$

It defines (m_i N m_j) when

$$F^1(m_i) < F^1(m_j),\ F^2(m_i) < F^2(m_j)$$
$$\text{and } F^3(m_i) < F^3(m_j)$$

It defines (m_i T m_j) when

$$F^1(m_i) < F^1(m_j),\ F^2(m_i) > F^2(m_j)$$
$$\text{and } F^3(m_i) > F^3(m_j)$$

In case of sequence quintuple, it consists of five sequences Γ^i, where $1 \le i \le 5$. Sequence quintuple generates a finite solution space which includes at least one optimal solution of 3D packing for volume optimization by decoding. Sequence quintuple defines (m_i W m_j) when

$$F^1(m_i) < F^1(m_j) \text{ and } F^2(m_i) < F^2(m_j)$$

It defines (m_i N m_j) when

$$F^3(m_i) < F^3(m_j) \text{ and } F^4(m_i) < F^4(m_j)$$

It defines (m_i T m_j) when

$$F^5(m_i) < F^5(m_j)$$

where m_i and m_j is overlapping in the projected xy-plane after WE- and NS- decoding.

6.2 Experiment of Packing Optimization

For fair comparison, all algorithms are implemented in Python environment on 2.16GHz PC with 3.00GB memory. A set of experiments was implemented by using the proposed ASA_X, in comparison to traditional SA and ASA. We are using MCNC, ami49_X and ami98_3D benchmarks. The ami49_X is produced by duplicating ami49 circuit X times. The ami98_3D is produced by inheriting the height and width of ami49_2, which is produced by duplicating ami49 circuit twice, and randomly getting the length between the given minimum and maximum dimensions. Based on the experiment using MCNC, ami49_X and ami98_3D benchmarks, the computational performance is considerably improved. The maximum computational time is within 14,400s (4 hours) each time. In the case of area minimization, the results gotten by the proposed ASA_X are normally better than the published data of 2D packing. In the case of volume minimization for 3D packing, the results gotten by the proposed ASA

are better than the data of traditional ASA and SA.

6.2.1 Cost Function

The area estimation is given by the minimum bounding rectangle including all modules, which is the total height *H* multiplied by the total width *W*. In practical implementation, we use a relative value as the cost function of area, i.e. the bounding area divided by the area of total modules, because any value with unit would not be scalable to use the experiments by diverse benchmarks.

The volume estimation is given by the minimum bounding rectangular parallelepiped including all modules, which is the total height H multiplied by the total width W and multiplied by the total length L. In the real implementation, the cost function (C) is using the relative value of volume, which is the volume of the minimum bounding box divided by the total volume of all modules.

6.2.2 Area Minimization

Table 26: Area optimization by ASA_X for 2D packing with SP representation

Benchmarks	Best (mm^2)	Average (mm^2)	Worst (mm^2)	Runtime (s)
apte	46.92	47.33	47.61	2.9
xerox	19.8	20.48	21.19	1.1
hp	8.95	9.17	9.31	1.5
ami33	1.18	1.23	1.28	15
ami49	36.67	37.68	38.69	29
ami49_2	73.01	75.17	77.12	121
ami49_4	146.4	150.3	155.1	519

For the area minimization of 2D packing, the best, average and worst cases of area minimization among 50 trials are gotten within 10 minutes each time using

MCNC and ami49_X benchmark by SP representation, as shown in Table 26. In the best case of area minimization, the results gotten by ASA_X are normally better than the published data. The comparison of solution and runtime between ASA and ASA_X on average is as shown in Table 27. The ASA_X reduced near 20% runtime with better solution. A near log-linear trend of average improvement rates from ASA to ASA_X is gotten. That means ASA_X should be more suitable for the packing problem with a larger number of modules.

Table 27: Average improvement of 2D packing with SP representation

Benchmarks	Solution(mm^2)		Runtime (s)		Improvement (%)	
	SA	ASA_X	SA	ASA_X	Solution	Runtime
apte	47.38	47.33	3.6	2.9	0.11%	19%
xerox	20.51	20.48	2.0	1.1	0.16%	45%
hp	9.18	9.17	2.8	1.5	0.11%	46%
ami33	1.24	1.23	19	15	0.86%	21%
ami49	37.96	37.68	42	29	0.79%	31%
ami49_2	75.98	75.17	189	121	1.15%	36%
ami49_4	152.2	150.3	712	519	1.37%	27%

6.2.3 Volume Minimization

As shown in Figure 38, it is the comparison of volume minimization using ST representation. The proposed ASA_X outperforms SA with the improvement of 3D packing ratio between 5% and 16%. The average improvement from SA to ASA is near 12%. The ASA_X outperforms ASA with the improvement between 2% and 8%. The average improvement of volume minimization from ASA to ASA_X is near 6%.

As shown in Figure 39, it is the comparison of volume minimization using SQ representation. The proposed ASA_X outperforms SA with the improvement of 3D packing ratio between 10% and 30%. The average improvement from SA to ASA is near 19%. The ASA_X outperforms ASA with the improvement between 4% and 20%.

The average improvement of volume minimization from ASA to ASA_X is near 10%.

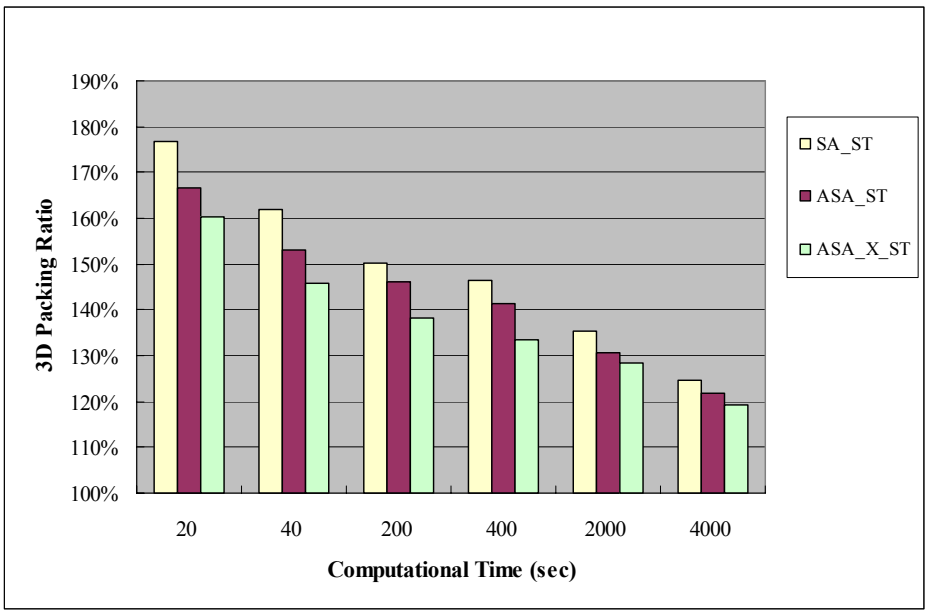

Figure 38: Comparison of computational performance among SA, ASA and ASA_X using ST representation

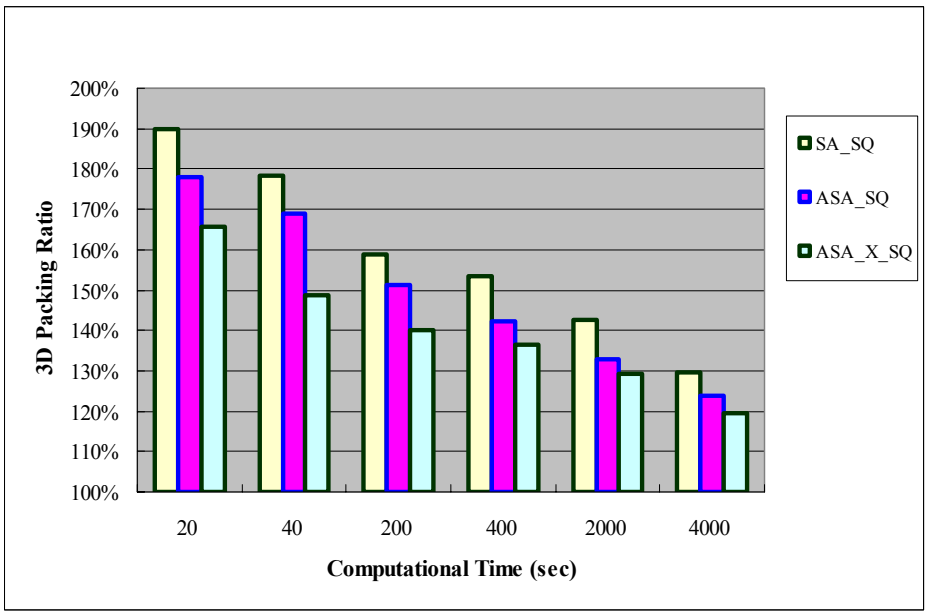

Figure 39: Comparison of computational performance among SA, ASA and ASA_X using SQ representation

6.3 Experiment of Placement Optimization

To confirm the computational performance of RRA, 2D rectangular placement problem in physical design is solved. Placement optimization [40-49] is a typical NP-hard problem with conflicting objectives. 2D rectangular placement is to position modules or blocks into a fixed rectangular shape with interconnect optimization and area minimization. To search near-optimal solutions efficiently, many researches explored representations for 2D placement, such as sequence pair (SP), BSG, O-tree [36], B*-tree [38], CBL [39], FAST-SP [42], Q-sequence [43], Selected SP [46], etc. A good representation should include at least one optimal solution with relatively small solution space. SP is used in this research since SP includes at least one optimal solution with regard to area minimization.

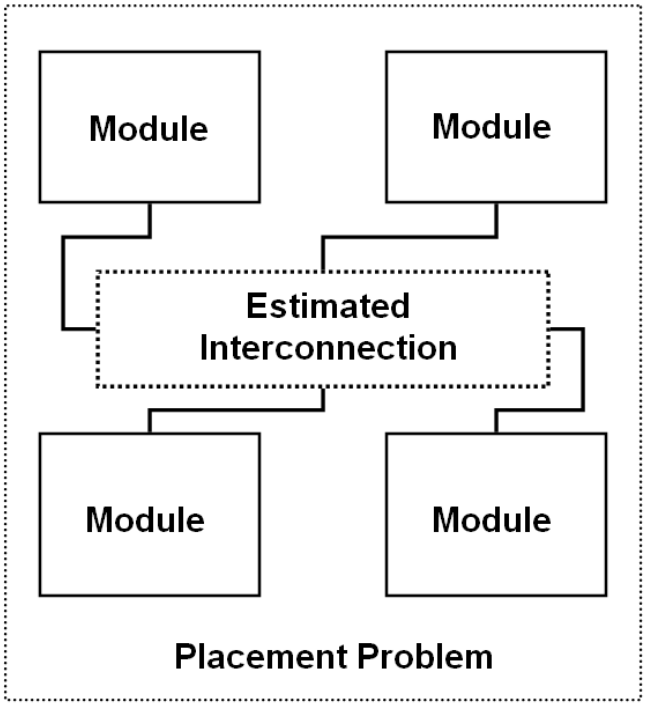

Figure 40: A placement optimization problem in physical design.

A placement optimization problem in physical design is as shown in Figure 40. The input is a set of modules and a set of nets which connect between modules. The objective and constraint is to optimize all expected performaces including power, delay and dimensions such that no modules are overlapping and enough space is left

to complete the interconnection between modules. The output is the location and the rotation of modules with the estimated interconnection.

2D rectangular placement problem is regarded as 2D rectangular packing with interconnect optimization. In case of sequence triple for 3D placement, it consists of three independent sequences Γ^i, where $1 \le i \le 3$. The coding and the decoding are based on TB-, NS- and WE- relation corresponding to the order of modules. For a given packing with n modules, the solution space is $(n!)^3$. If the rotation of the module is not fixed, then the solution space will increase to $(n!)^3 6^n$. In case of sequence quintuple for 3D placement, it consists of five sequences Γ^i, where $1 \le i \le 5$. Sequence quintuple generates a finite solution space which includes at least one optimal solution of 3D packing for volume optimization by decoding. For a given packing with n modules, the solution space is $(n!)^5$. If the rotation of the module is not fixed, then the solution space will increase to $(n!)^5 6^n$.

For a fair comparison, all algorithms to solve 2D placement problem are implemented in Python environment on 2.16GHz PC with 3.00GB memory. With regard to ami49_X and MCNC benchmarks, a set of experiments was implemented. The ami49_X is produced by duplicating ami49 with *X* times. It has the scalable number of blocks (49**X*) and nets (408**X*). To compare with the published results fairly, MCNC benchmarks are used for area minimization.

6.3.1 Cost Function

2D placement problem includes three objectives, including area, power and delay. Since the power and the delay are conflicting, we have to get Pareto improvement of multi-objective optimization. Let us define the total cost function as following equation.

$$C_t = \alpha \cdot C_p + \beta \cdot C_d + \gamma \cdot C_a$$

Where α, β and γ are user-defined to satisfy $\alpha+\beta+\gamma=1$. The total cost function (C_t) is made up of the cost of power (C_p), the cost of delay (C_d) and the cost of area (C_a).

For power estimation, the dynamic power of a net n_i is proportional to $C(n_i)$, $V_{dd}(n_i)^2$, $f(n_i)$ and $S(n_i)$, where $C(n_i)$ is the capacitance of the net n_i, $V_{dd}(n_i)$ is the voltage of power supply, $f(n_i)$ is the clock frequency, and $S(n_i)$ is switching probability of the net. The relative value, which is the power divided by the limit of lowest power, is used as the value of C_p because the relative values are scalable. The $C(n_i)$ is proportional to the length of net, and we use Len_i to represent its value. We assume that $V_{dd}(n_i)$ and $f(n_i)$ are same for each net and $S(n_i)$ is randomly defined between 0 and 1 in case of no specific information. The power is simplified as the function of Len_i and $S(n_i)$. For delay estimation, the maximal delay among all nets is used. The relative value, which is the maximal delay divided by the limit of shortest wire length of nets, is used as the value of C_d. To get the wire length estimation of each net, the half perimeter wire length (*HPWL*) is used. The area estimation is given by the minimum bounding rectangle including all modules. The relative value, which is the bounding area divided by the area of total modules, is used as the value of C_a.

6.3.2 Area Minimization

Table 28: Area minimization using SA and RRA (within 12 min)

MCNC	Best (mm^2)		Average (mm^2)		Worst (mm^2)	
	SA	RRA	SA	RRA	SA	RRA
apte	47.08	46.92	47.38	47.29	47.71	47.54
xerox	19.8	19.8	20.51	20.44	21.27	21.11
hp	9.03	8.95	9.18	9.14	9.38	9.25
ami33	1.19	1.18	1.24	1.22	1.30	1.27
ami49	37.23	36.52	37.96	37.17	38.97	38.01
ami49_2	74.13	72.99	75.98	74.46	77.65	75.78
ami49_4	149.2	145.8	152.2	148.4	156.7	150.7

Table 29: Average area improvement from SA to RRA

MCNC	Solution		Runtime		Improvement (%)	
	SA	RRA	SA	RRA	Solution	Runtime
apte	47.38	47.29	4.1	1.6	0.19%	61.0%
xerox	20.51	20.44	1.9	0.8	0.36%	57.9%
hp	9.18	9.14	2.7	1.1	0.45%	59.3%
ami33	1.24	1.22	22	13	1.73%	40.9%
ami49	37.96	37.17	45	26	2.23%	42.2%
ami49_2	75.98	74.46	194	91	2.15%	53.1%
ami49_4	152.2	148.4	720	371	2.70%	48.5%

Table 30: Comparison between RRA and the latest ASA

MCNC	Solution		Runtime		Improvement (%)	
	ASA	RRA	ASA	RRA	Solution	Runtime
apte	47.33	47.29	2.9	1.6	0.09%	44.8%
xerox	20.48	20.44	1.1	0.8	0.21%	27.3%
hp	9.17	9.14	1.5	1.1	0.34%	26.7%
ami33	1.23	1.22	15	13	0.86%	13.3%
ami49	37.68	37.17	29	26	1.44%	10.3%
ami49_2	75.17	74.46	121	91	1.00%	24.8%
ami49_4	150.3	148.4	519	371	1.34%	28.5%

In case of area optimization, let γ be 1 and $\alpha+\beta$ be 0. As shown in Table 28, the best, average and worst cases among 50 trials are gotten within 12 minutes. As shown in Table 29, the specific improvement of RRA is from 0.19% to 2.70%. RRA reduced at least 40% runtime with better solution by comparison to SA with same

representation. As shown in Table 30, the comparison between RRA and the latest proposed ASA shows the considerable improvement of both solution and runtime within short runtime. However, when the runtime is increased to 15 minutes, the average results of ASA_X are better than those of RRA as shown in Table 31.

Table 31: Comparison between RRA and ASA_X (5 Trials, Runtime: 15 min)

MCNC	Solution		Runtime (s)		Improvement (%)	
	ASA_X	RRA	ASA_X	RRA	Solution	Runtime
apte	46.93	47.12	900	900	-0.40%	0%
xerox	19.8	19.9	900	900	-0.51%	0%
hp	8.96	9.01	900	900	-0.56%	0%
ami33	1.18	1.19	900	900	-0.85%	0%
ami49	36.68	36.81	900	900	-0.35%	0%
ami49_2	73.02	73.19	900	900	-0.23%	0%
ami49_4	146.9	147.1	900	900	-0.14%	0%

6.3.3 Interconnect Optimization

In the case of interconnect optimization, let γ be 0 and $\alpha+\beta$ be 1. To get the Pareto frontiers of two conflicting objectives, α is randomly produced from 0.1 to 0.9, and the relative value is divided by the limit of optima, then we pick up 240 solutions for comparison. We are using ami49_4 benchmark for comparison this time. The experimental data of SA within 30 minutes are shown in Figure 41, and the results of RRA within the same runtime are shown in Figure 42.

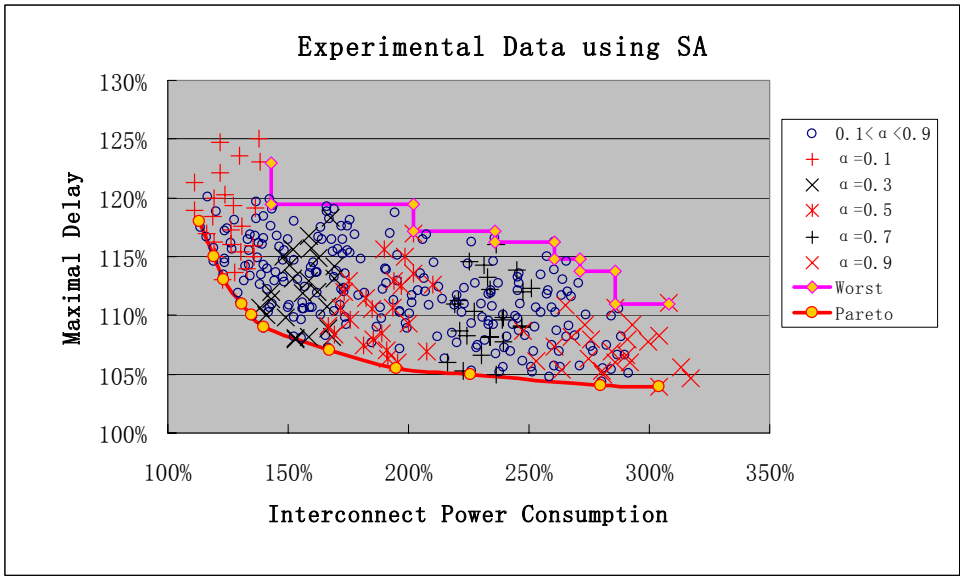

Figure 41: The best and worst cases of experimental data using SA

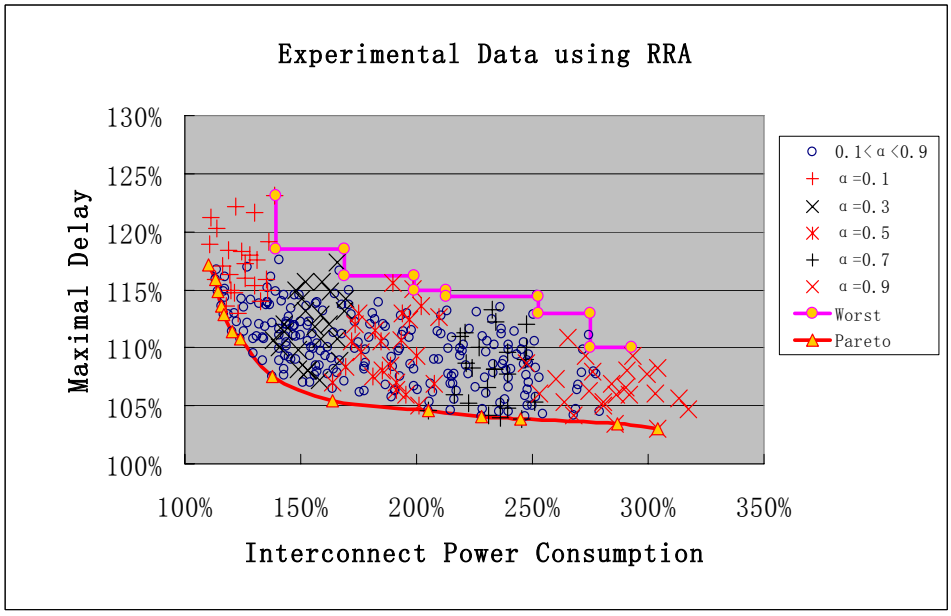

Figure 42: The best and worst cases of experimental data using RRA

6.3.4 Pareto Improvement

The overall Pareto improvement which is more than 30% improvement of interconnect power consumption with no degradation of performance is obtained. The

Pareto improvement means the improvement of Pareto frontier between two conflicting objectives as shown in Figure 43. The "overall" means that Pareto improvement is achieved in all tested range of objectives, such as power consumptions from 100% to 350%, by multiple runs of RRA with various different choices of weighting coefficients. As shown in Figure 44, it shows that RRA obtains at least 29% Pareto improvement with the constraint of less than 107.5% maximal delay. To get the worst cases, we tested more 120 solutions with α given by 0.1, 0.3, 0.5, 0.7, and 0.9. As a result, RRA got near 24% worst-case mitigation on average for power consumption with no degradation of maximal delay as shown in Figure 45. For all tested ami49_X with X from 1 to 12, block number from 49 to 588, and net number from 408 to 4896, the similarly improved results are gotten.

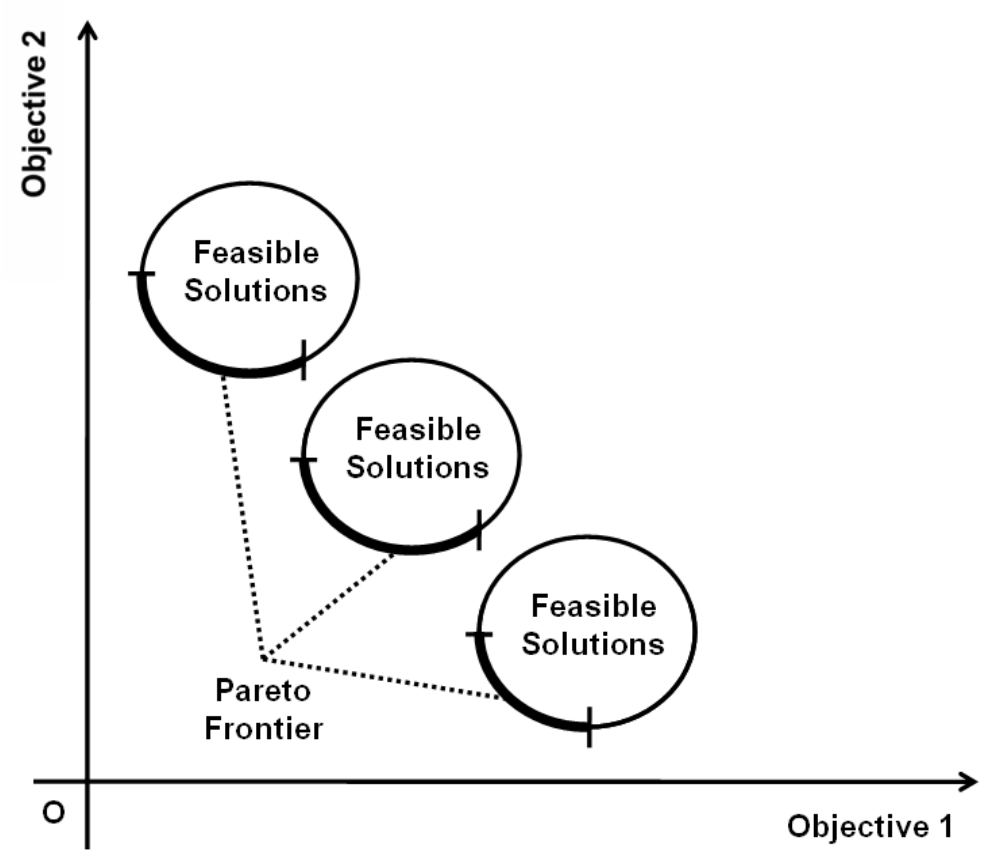

Figure 43: Pareto frontier of conflicting objectives

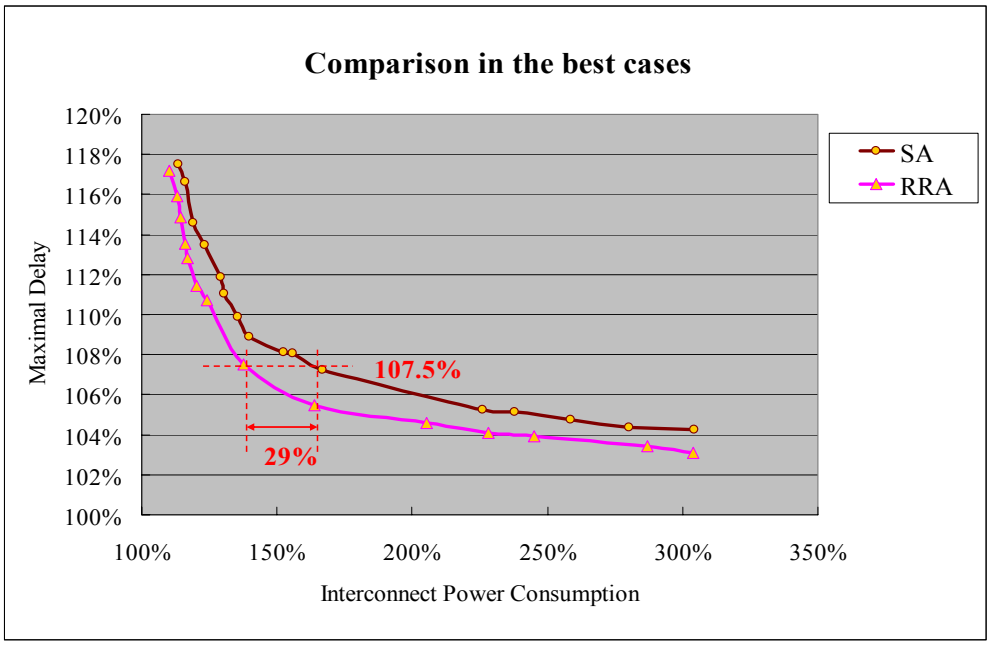

Figure 44: Pareto improvement from SA to RRA

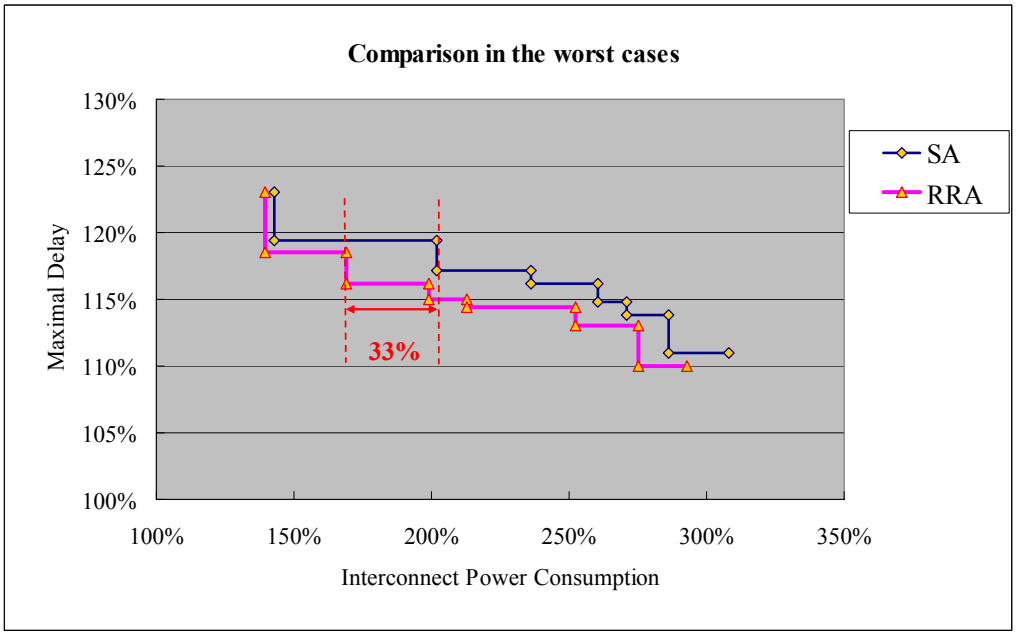

Figure 45: Worst-case improvement from SA to RRA

6.4 Summary

To confirm the computational efficiency empirically, packing and placement problems in 2D/3D IC physical design optimization are solved and improved. Based

on the experimental data using MCNC, ami49_X and ami98_3D benchmarks, the computational performance is considerably improved. Moreover, based on the experimental data of placement, the computational performance is considerably improved. The overall Pareto improvement with conflicting objectives is obtained by using the proposed RRA. In the worst cases, lower power design with no degradation of maximal delay is gotten by using the proposed RRA.

7. Conclusion and Future Works

Heuristics [60-66] are widely used to get near-optimal solution of NP-hard problem in product design optimization by exploring the solution space. However, with the rapid advancement of electronic products, it is increasingly hard to satisfy high-performance requirements of problem solving and product design due to short life cycle and high complexity. Especially, the efficiency of emerging search algorithms is limited due to the following reason. A local optimal solution which is similar to an obtained local optimal solution would not be explored. It is expected that a better local optimal solution exists which is similar to an obtained local optimal solution.

To improve the existing search methods, various high-performance techniques including moving methods, adaptive selection and similar optimum search are discussed. A new variation of simulated annealing, named adaptive simulated annealing with crossover (ASA_X), is proposed. A guide with adaptive probabilities is used to select moving methods including a special crossover. Moreover, as a novel high-performance heuristic, relay-race algorithm (RRA) is proposed to approach a global optimum which is the best of all local optimal solutions by exploring similar local optimal solutions more efficiently within shorter runtime for NP-hard problem. RRA includes three stages: rough search, focusing search and relay. The rough search is designed to get over small hills on the solution space and to approach a local optimal solution as fast as possible. The focusing search is used to reach a local optimum as close as possible. The relay is designed to escape the local optimal solution in only one step and to maintain search continuity simultaneously.

To confirm the efficiency of the proposed methods empirically, the packing and placement problems in 2D/3D IC physical design are solved by ASA_X and RRA as the typical applications of product design optimization. Based on the experimental data using MCNC, ami49_X and ami98_3D benchmarks, the computational performance is considerably improved. In the case of area minimization for 2D

packing, the results gotten by ASA_X are normally better than the published data within a short runtime. In the case of volume minimization for 3D packing, the results gotten by ASA_X are also better than the data of existing methods. Since multi-objective problems have become common in practice nowadays, the Pareto improvement of one objective with no degradation of another objective is one of the biggest challenges of problem solving. By using the proposed RRA, the overall Pareto improvement of placement with conflicting objectives is obtained. In the worst cases, the big average improvement of power is obtained by the proposed RRA without any degradation of maximal delay.

With regard to future work, similar optimum search based heuristics have potential to solve and improve more NP-hard problems. For example, RRA has potential to solve 3D placement optimization prolem with high performace. Population based RRA using parallel computation is a direct extension of the current RRA to improve the global search ability and the search stability in future. Fuzzy RRA which is based on fuzzy sets should be a potential extension to have more applications to improve the existing methods. Besides, diverse optimum search which focusing on diverse local optimal solutions to approach global optimum would be proposed as a novel optimization technique. In the near future, multi-level placement, as a simplified model of system design, would be proposed with applications to both general physical design and system design.

Besides, the single-objective function problem is solved multiple times independently with fixed values of coefficients to obtain the Pareto frontiers of conflicting objectives in this research. Furthermore, the random values of coefficients are used to confirm the best case and the worst case. Based on the experimental data, the proposed RRA is more efficient to get Pareto optimal solutions comparing with SA under the same conditions. However, the local optimal solutions searched by iterative improvement methods might be limited in the case that coefficients are fixed during search. Changing the coefficient during search in RRA to improve the search efficiency is one of our future works. In short, the final goal of this research as well as its future work is to find better methods, to solve harder problems and to realize more

applications.

References

[1] J. W. Escobara, R. Linfatia and P. Totha, “A Two-Phase Hybrid Heuristic Algorithm for the Capacitated Location-Routing Problem,” Computers & Operations Research, Vol. 40, pp. 70-79, 2013.

[2] A. Sakalli, E. Yesil, E. Musaoglu, C. Ozturk and M. F. Dodurka, , “Heuristic bubble algorithm for a linehaul routing problem: An extension of a vehicle routing problem with pickup and delivery,” Proceedings of IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI), pp. 435-439, 2013.

[3] L. Guo, S. Zhao, S. Shen and C. Jiang, “Task Scheduling Optimization in Cloud Computing Based on Heuristic Algorithm,” Journal of Networks, Vol. 7, pp. 547-553, 2012.

[4] A. Kaveh and S. Talatahari, “A Novel Heuristic Optimization Method: Charged System Search,” Journal of Acta Mechanica, Vol. 213, pp. 267-289, 2010.

[5] G. Hwang, F. Kuo, P. Yin and K. Chuang, “A Heuristic Algorithm for Planning Personalized Learning Paths for Context-aware Ubiquitous Learning,” Computers & Education, Vol. 54, pp. 404–415, 2010.

[6] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan and J. R. Woodward, “A Classification of Hyper-heuristic Approaches,” Handbook of Metaheuristics, International Series in Operations Research & Management Science, Vol. 146, pp. 449-468, 2010.

[7] F. Xhafa and A. Abraham, “Computational models and heuristic methods for Grid scheduling problems,” Future Generation Computer Systems, Vol. 26, pp. 608–621, 2010.

[8] A. Kaveh and S. Talatahari, “A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments,” Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 400-407, 2010.

[9] J. Hoffmann and B. Nebel, “The FF Planning System: Fast Plan Generation Through Heuristic Search,” Journal Of Artificial Intelligence Research, Vol. 14, pp. 253-302, 2001.

[10] R. D. Shachter, D. M. Eddy, V. Hasselblad and R. Wolpert, “A Heuristic Bayesian Approach to Knowledge Acquisition: Application to Analysis of Tissue-Type Plasminogen Activator,” Proceedings of the Third Conference on Uncertainty in Artificial Intelligence, pp. 229-236, 1987.

[11] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, “Optimization by Simulated Annealing,” Science, Vol. 220, No. 4598, pp. 671–680, 1983.

[12] E. H. L. Aarts, J. Korst, “Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing,” John Wiley & Sons, 1989.

[13] S. Y. Ho, Y. K. Lin and W.C. Chu, “An orthogonal simulated annealing algorithm for large floorplanning problems,” IEEE Transactions on VLSI Systems, Vol. 12, No. 8, pp. 874-877, 2004.

[14] P. Subbaraj, S. Sankar and S. Anand, “Parallel Genetic Algorithm for VLSI Standard Cell Placement,” Proceedings of International Conference on Advances in Computing, Control, and Telecommunication Technologies, pp. 70-84, 2009.

[15] D. W. Philip, “Neural Computing: Theory and Practice,” Van Nostrand Reinhold, 1989.
[16] J. Kennedy and R. Eberhart, “Particle swarm optimization,” Proceedings of the 4th IEEE International Conference on Neural Networks, pp. 1942-1948, 1995.
[17] P. Hansen and N. Mladenovic, “Variable neighborhood search: Principles and applications,” European Journal of Operational Research, Vol. 130, Issue 3, pp. 449-467, 2001.
[18] P. N. Suganthan, “Particle swarm optimizer with neighborhood operator,” Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1958-1962, 1999.
[19] K. Parsopoulos and M. Vrahatis, “Recent approaches to global optimization problems through particle swarm optimization,” Natural Computing, Vol 40, No. 1, pp. 235-306, 2002.
[20] C. H. Papadimitriou and K. Steiglitz, “Combinatorial Optimization: Algorithms and Complexity,” Dover Publications, 1998.
[21] M. Avriel, M.J. Rijckaert and D.J. Wilde, “Optimization and Design,” Prentice-Hall, 1973.
[22] M. Avriel and R.S. Dembo, “Mathematical Programming Studies on Engineering Optimization,” North-Holland, 1979.
[23] J. N. Siddall, “Optimal Engineering Design,” CRC, 1982.
[24] E.J. Cramer, J. E. Dennis Jr., P.D. Frank, R.M. Lewis and G.R. Shubin, “Problem Formulation for Multidisciplinary Optimization,” SIAM J. Optim., Vol 4, No. 4, pp754-776, 1994.
[25] M. Achille, “Physical programming - Effective optimization for computational design,” AIAA Journal, Vol. 34, No. 1, pp. 149-158, 1996.
[26] J. Sobieszczanski-Sobieski and R. T. Haftka, “Multidisciplinary aerospace design optimization: survey of recent developments,” Structural optimization, Vol. 14, Issue 1, pp 1-23, 1997.
[27] D. Assanis, et al., “Optimization Approach to Hybrid Electric Propulsion System Design,” Mechanics of Structures and Machines, Vol. 27, Issue 4, pp 393-421, 1999.
[28] G. J. Kacprzynski, M. J. Roemer and A. J. Hess, “Health management system design: Development, simulation and cost/benefit optimization,” Proceedings of IEEE Aerospace Conference, Vol. 6, pp 3065-3072, 2002.
[29] G. N. Vanderplaats, “Multidiscipline Design Optimization,” Vanderplaatz R&D, Inc., 2007.
[30] M. Bob, “Sound Systems: Design and Optimization: Modern Techniques and Tools for Sound System Design and Alignment, ” ISBN 9780240521565, Focal Press, 2nd edition, 2009.
[31] D. F. Wong and C. L. Liu, “A New Algorithm for Floorplan Design,” Proceedings of Design Automation Conference, pp. 101–107, 1986.
[32] R. H. J. M. Otten, “Automatic Floorplan Design,” Proceedings of Design Automation Conference, pp. 261-267, 1982.
[33] T. C. Wang, and D. F. Wong, “An Optimal Algorithm for Floorplan Area Optimization,” Proceedings of Design Automation Conference, pp. 180-186, 1990.
[34] S. Nakatake, K. Sakanushi, Y. Kajitani and M. Kawakita, “The Channeled BSG: A Universal Floorlan for Simutaneous Place/Route with IC Application,” Proceedings of International Conference on CAD, pp. 418-425, 1998.
[35] P. S. Dasgupta, S. Sur-Kolay, and B. B. Bhattacharya, “A Unified Approach to Topology Generation and Optimal Sizing of Floorplans,” IEEE Transactions on

Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No.2, pp. 126–1335, 1998.
[36] P. N. Guo, C. K. Cheng and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," Proceedings of Design Automation Conference, pp. 268-273, 1999.
[37] T. Takahashi, "A New Encoding Scheme for Rectangle Packing Problem," Proceedings of IEEE Asia South Pacific Design Automation Conference, pp. 175–178, 2000.
[38] Y. C. Chang, Y. W. Chang, G. M. Wu and S. W. Wu, "B*-trees: a new representation for non-slicing floorplans," Proceedings of Design Automation Conference, pp. 458-463, 2000.
[39] X. Hong, G. Huang, Y. Cai, S. Dong, C. K. Cheng and J. Gu, "Corner Block List: An effective and efficient topological representation of non-slicing floorplan," Proceedings of the International Conference on Computer Aided Design, pp. 8-12, 2000.
[40] M. Tsuboi, C. Kodama, K. Fujiyoshi, K. Sakanushi, and A. Takahashi, "Linear Time Decodable Rectangular Dissection to Represent Arbitrary Packing using Q-Sequence," The 10th Workshop on Synthesis And System Integration of Mixed Information technologies, pp. 272–278, 2001.
[41] S. Nakatale, Y. Kudo and Y. Kajitani, "Consistent Floorplanning with Hierarchical Superconstraints," IEEE Transactions on CAD of IC and Systems, Vol. 21, No. 1, pp. 42-49, 2001.
[42] X. Tang and D. F. Wong, "FAST-SP: a fast algorithm for block placement based on sequence pair," Proceedings of IEEE Asia South Pacific Design Automation Conference, pp. 521-526, 2001.
[43] C. Zhuang, K. Sakanushi, L. Jin and Y. Kajitani, "An enhanced Q-sequence augmented with empty-room-insertion and parenthesis trees," Proceedings of Design, Automation and Test in Europe Conference and Exhibition, pp. 61-68, 2002
[44] Y. Kudo, S. Nakatale, Y. Kajitani and M. Kawakita, "Explicit Expression and Simultaneous Optimization of Placement and Routing," Proceedings of IEEE Asia South Pacific Design Automation Conference, pp. 467-472, 2002.
[45] Y. Kudo, S. Nakatale, Y. Kajitani and M. Kawakita, "Chip Size Estimation Based on Wiring Area," Proceedings of IEEE Asia South Pacific Conference on Circuit and Systems, Vol 2, pp. 113-118, 2002.
[46] C. Kodama and K. Fujiyoshi, "Selected sequence-pair: an efficient decodable packing representation in linear time using sequence-pair," Proceedings of IEEE Asia South Pacific Design Automation Conference, pp. 331-337, 2003.
[47] Y. Kohira, C. Kodama, K. Fujiyoshi, A. Takahashi, "Evaluation of 3D-Packing Representations for Scheduling of Dynamically Reconfigurable Systems," Proceedings of IEEE International Symposium on Circuits and Systems, pp. 4487-4490, 2006.
[48] M. Tang and X. Yao, "A Memetic Algorithm for VLSI Floorplanning," IEEE Transactions on Systems, Man and Cybernetics, Vol. 37, No. 1, pp. 62-69, 2007.
[49] F. Mao, N. Xu and Y. Ma, "Hybrid Algorithm for Floorplanning Using B*-tree Representation," Proceedings of Third International Symposium on Intelligent Information Technology Application, pp. 228-231, 2009.
[50] H. Onodera, Y. Taniguchi, K. Tamaru, "Branch-and-Bound Placement for Building Block Layout," Proceedings of Design Automation Conference, pp. 433-439, 1991.

[51] B. T. Preas and W. M. VanCleemput, “Placement Algorithms for Arbitrarily Shaped Blocks,” Proceedings of Design Automation Conference, pp. 474-480, 1979.
[52] M. Igusa, M. Beardslee and A.S. Vincentelli, “ORCA: A Sea-of Gates Place and Route System,” Proceedings of Design Automation Conference, pp. 122-127, 1989.
[53] J. M. Kleinhans, G. Sigl, F.M. Johannes, etal, “GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 10, No. 3, pp. 356-365, 1991.
[54] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, “Module placement on BSG-structure and IC layout applications,” Proceedings of International Conference on CAD, pp. 484-491, 1996.
[55] 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, 1996.
[56] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh and C.K. Wong, “Provably Good Performance-Driven Global Routing,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 11, No. 6, pp. 739-752, 1992.
[57] J. Cong, T. Kong, J. Shinnerl, M. Xie and X. Yuan, “Large Scale Circuit Placement,” ACM Transaction on Design Automation of Electronic Systems, Vol. 10, No. 2, pp. 389-430, 2005.
[58] A. Drakidis, R. J. Mack and R. E. Massara, “Packing-based VLSI module placement using genetic algorithm with sequence-pair representation,” Proceedings of IEE on Circuits, Devices and Systems, pp. 545-551, 2006.
[59] B. A. Rosdi and A. Takahashi, “Replacement of Register with Delay Element for Reducing the Area of Pipelined Circuits,” Proceedings of IEEE Asia-Pacific Conference on Circuits and Systems, pp. 802-805, 2006.
[60] D. Kahneman, A. Tversky, and P. Slovic, “Judgment under Uncertainty: Heuristics & Biases,” Cambridge, UK, Cambridge University Press, ISBN 0-521-28414-7 , 1982.
[61] G. Gigerenzer, P. M. Todd, and the ABC Research Group, “Simple Heuristics That Make Us Smart,” Oxford, UK, Oxford University Press. ISBN 0-19-514381-7, 1999.
[62] Olga Kiss, “Heuristic, Methodology or Logic of Discovery: Lakatos on Patterns of Thinking,” Science, Vol. 220, No. 4598, pp. 671–680, 2006.
[63] R. Poli, J. Kennedy and T. Blackwell, “Particle swarm optimization: An overview,” Swarm Intelligence, Vol. 1, No. 1, pp. 33-57, 2007.
[64] Gerd Gigerenzer and Christoph Engel, “Heuristics and the Law,” Cambridge, The MIT Press, ISBN 978-0-262-07275-5, 2007.
[65] S. Bandyopadhyay, S. Saha, U. Maulik and K. Deb, "A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA", IEEE Transactions on Evolutionary Computation, Vol. 12, No.3, pp.269-283, 2008.
[66] G. Gigerenzer and W. Gaissmaier, “Heuristic Decision Making,” Annual Review of Psychology. Vol. 62. pp. 451-482, 2011.

Publications

Journal Paper and Book

[1] Y. Sheng and A. Takahashi, "A Novel High-Performance Heuristic Algorithm with Application to Physical Design Optimization," IEICE Transaction on Fundamentals of Electronics, Communications and Computer Sciences, Acceptance with Minor Revision, 2014.
[2] Y. Sheng and A. Takahashi, "A New Variation of Adaptive Simulated Annealing for 2D/3D Packing Optimization," Information Processing Society of Japan (IPSJ) Transactions on System LSI Design Methodology, Vol.6, pp. 94-100, 2013.
[3] Y. Sheng and A. Takahashi, "A Simulated Annealing Based Approach to Integrated Circuit Layout Design," Simulated Annealing - Single and Multiple Objective Problems, InTech, ISBN 978-953-51-0767-5, pp. 239-260, 2012.

International Conference

[1] Y. Sheng, A. Takahashi and S. Ueno, "2-Stage Simulated Annealing with Crossover Operator for 3D-Packing Volume Minimization," Proceedings of The 17th Workshop on Synthesis And System Integration of Mixed Information technologies, pp. 227-232, 2012.
[2] Y. Sheng, A. Takahashi and S. Ueno, "Relay-Race Algorithm: A Novel Heuristic Approach to VLSI/PCB Placement," Proceedings of IEEE Computer Society Annual Symposium on VLSI, pp. 96-101, 2011.
[3] Y. Sheng, A. Takahashi and S. Ueno, "RRA-Based Multi-Objective Optimization to Mitigate the Worst Cases of Placement," Proceedings of The IEEE 9th International Conference on ASIC, pp. 357-360, 2011.

Technical Report in Japan

[1] Y. Sheng, A. Takahashi and S. Ueno, "A Stochastic Optimization Method to Solve General Placement Problem Effectively," Information Processing Society of Japan (IPSJ) DA Symposium, Vol. 2011, No.5, pp. 27-32, 2011.
[2] Y. Sheng, A. Takahashi and S. Ueno, "An Improved Simulated Annealing for 3D Packing with Sequence Triple and Quintuple Representations," IEICE Technical Report on VLSI Design Technologies, Vol.111, No.324, pp.209-214, 2011.
[3] Y. Sheng, A. Takahashi and S. Ueno, "MSA: Mixed Stochastic Algorithm for Placement with Larger Solution Space," IEICE Technical Report on VLSI Design Technologies, Vol.111, No.216, pp.11-16, 2011.