Meta-Memes for Combinatorial Optimization

Song Li Qin

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirement for the degree of

Doctor of Philosophy

2011

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

_____ _____

DATE Song Li Qin

Acknowledgements

I would like to thank my supervisor, Dr Lim Meng Hiot, for his supervision and guidance, specially the patience to me. In addition to his concern about my research work, I am very grateful for his cares on my life. His style of supervision and the way of communication will definitely affect my future.

I would also like to thank Dr Ong Yew Soon for his help and guidance, acting as a good friend and co-supervisor of me. I also want to express my appreciation to Dr Ponnuthurai Nagaratnam Suganthan for his friendly assistance and guidance.

Most importantly, special thanks go to my family, especially my wife, thanks for their understanding and support. I also like to thank Mdm. Noraidillah Bte Mohamed Ali from Intelligent System Centre for her continuous kindly help.

Abstract

This research aims at establishing a framework for managing and coordinating optimization methods within a search algorithm. Each optimization method is viewed as a meme. The meme is therefore an algorithmic abstraction that dictates how solution search is carried out. At a higher level, the configuration of the entire search algorithm is dictated by a meta-meme. In this case, a meta-meme can be perceived as being a recipe that provides specifications on how the strategies or memes are coordinated during a search.

This work uses a neural metaphor to capture the essence of meta-memes. Through a framework known as neural meta-memes framework (NMMF), the coordination of memes during a search can be established based on the instances of earlier problem-solving sessions. Specifically, smaller or simple instances of problems to be solved are used as the basis to train the neural network using back-propagation update rule. The trained neural network capturing the experience acquired through earlier problem-solving sessions provides linkages between problems and memes, significantly enhances the search capability when dealing with complex or large-scale problems. Additionally, a method is further established with which memes in the form of patterns or solution fragments can serve as a basis for generating potentially good quality initial

solutions to enhance the efficiency of a meta-meme during a search.

The entire scheme was validated on the quadratic assignment problems, arguably one of the hardest among all the combinatorial optimization problems. Experimental results achieved are highly competitive, the technique outperforming that of well-known published results. Although the efficacy of the scheme on combinatorial optimization problems has been shown, in principle the meta-meme approach can also be applied to continuous optimization. It is also worth noting that besides the neural meta-memes studied in this research, other well-established meta-memes manifestations such as fuzzy rules, graphs, heuristic rules, etc., are also potentially applicable.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Introduction

Combinatorial optimization problems involve minimization or maximization of discrete problems with finite number of feasible solutions. There are many real world applications for combinatorial optimization problems. Examples include facility layout, university curriculum timetabling, quadratic assignment problem (QAP), travelling salesman problem (TSP), vehicle routing problem (VRP), job-shop scheduling problem (JSSP), etc. Enumeration may seem to be a viable option in solving such problems since the number of solutions is limited. In theory, the optimum solution of a combinatorial optimization problem can be identified by exploring all possible solutions. In practice, however, the number of solutions increases exponentially with the increase in problem size. Exact enumerative methods are not practical when the size of a problem increases up to a certain level (except for some specially structured problems). Heuristic methods have been well studied in recent decades and the performance of such methods for large-scale combinatorial optimization problems has been exceptional. Although heuristic methods do not guarantee finding the optimum solution, they are able to produce solutions with

acceptable qualities in reasonable computational time period compared to deterministic methods. Within heuristic methods, metaheuristics are a class of methods designed to handle the search for optimum solution by iteratively improving the quality of the candidate solutions. Popular metaheuristics include genetic algorithms (GA), memetic algorithms (MA), simulated annealing (SA), tabu search (TS), ant colony optimization (ACO) and particle swarm optimization (PSO), etc.

A meme, by its original definition, is a unit of cultural information inheritable between individuals (Dawkins 1976). For example, it can be passed from a parent to its offspring or simply between interacting individuals. In the recent two decades, the concept of meme has gained acceptance rapidly with many new search methods proposed. In (Meuth et al. 2009), the meme's definition is extended, a meme being described as a mechanism that captures the essence of knowledge in the form of procedures that affect the transition of solutions during a search. Any optimization method encompasses search procedure(s) and routine(s), no matter how basic, bring some unique form of improvement to the search. This is usually not easily replaced by any other methods. In general, the behavior of a basic or hybrid search method is consistent in a static or controlled environment hence the search results tend to be predictable. In this sense, a search method can be modeled as a reliable knowledge system, in other words, a meme. Meanwhile, a hybridized method which manages or incorporates different methods collectively to achieve better performance can be modeled as a form of meta-meme method. In here, meta-memes refer to strategies for incorporating different memes

collectively.

1.2 Motivation

Different types of search methods are introduced and studied for various combinatorial optimization problems. Optimization methods which consist of two or more basic search methods or memes are probably the most efficient methods for complex combinatorial optimization problems.

Similar to biological evolution, hybridization or collaboration between different search methods or memes normally brings about greater robustness and higher efficiency. For most cases, the performance of a meta-meme method is not worse off than that of its constituent memes independently. With a meta-meme approach, different memes are arranged in serial orders and share the same search space. For example, within a typical hybrid genetic method, the genetic operators and local search operator carry out the search on the same individuals in one population. The memes cooperate with each other through their corresponding search results on the same search space. In order to retain the specialty of each meme during the collaboration of two or more independent memes, a scheme capable of incorporating multiple memes by providing each meme a private search space and also a communication protocol between the constituent memes is desirable.

Hybridization offers greater flexibility and robustness compared to the memes which rely on a single search method operating independently. Empirical studies show that hybrid or meta-meme methods are especially efficient in solving large-scale combinatorial optimization problems. The technique of manipulating two or more memes to work cooperatively and exploiting the positive traits of each of the memes is the most fundamental factor for the overall enhancement in the performance of a meta-meme method. Appropriate balancing of workload of the single memes is crucial for enhancing the performance of a meta-meme approach. Single memes within a meta-meme method should not be under- or over-exploited. For example, within a typical hybrid genetic method (Lim et al. 2002), the genetic operators are allocated relatively lighter workload compared to the local search method. The genetic operators are invoked only once every generation. The local search operator, on the other hand, allows multiple iterations in the same generation. Depending on the characteristics of the problems being solved, such gross imbalance may possibly lead to under utilization of one meme (the genetic algorithm) and over utilization of the other (the local search method). It is rational to suggest that a meta-meme approach which allocates each meme with sufficient leverage to search fully but not overly extends its exploitative capacity is more desirable. Furthermore, with most meta-meme approaches, the memes are usually arranged with certain predefined orders as well as fixed search steps. This has a tendency of succumbing to a static or steady-state enforcement of exploration and exploitation biases throughout the search. It is well-known that with complex combinatorial optimization problems, the unpredictability of the solution landscape creates a need for different

memes with different ways of balancing exploration and exploitation to achieve good and efficient search performance. For example, one group of problems may be better served if exploration is given greater emphasis while another group may find greater exploitation to be beneficial. Accordingly the "*no free lunch theorem*" (Wolpert and Macready 1997) has been generally accepted as a stipulation that there is no single method or approach which can consistently solve any type of combinatorial optimization problems. It follows that no single or fixed meta-meme method can consistently solve all combinatorial optimization problems. As such, it is desirable if a system is capable of dynamically generating different recipes of managing or coordinating different memes so as to achieve optimal performance for different problems. Based on the linkages between different combinatorial optimization problems and memes, and the linkages between different problems, the experience gained from earlier problem-solving sessions can be used to direct the construction of appropriate search methodologies for previously unsolved problems. This theoretical evidence supports the idea of establishing the dynamic and problem-adaptive system for combinatorial optimization problems.

Individual solutions for many combinatorial optimization problems are represented by permutations. Two or more elements of a solution permutation for a combinatorial optimization problem together can be treated as evolutionary replicators or memes just like biological genes on a chromosome. The concept of utilizing experience collected from earlier problem-solving sessions can be further extended to the utilization of existing solutions collected from earlier problem-solving sessions. In general, if the

information about the optimum solution of an unsolved problem is unknown in advance, it is a good choice to initialize a population with randomly generated permutations in the beginning of the search. In other words, random starting points are preferred with the unknown large search space. During a search process, a search method leads the random starting points to local optima or the global optimum. But if there is any prior knowledge on a problem before it is solved, the initial search points could be manipulated so that the search process can be faster or more efficient in uncovering the optimum. Another fact is that the search space of a problem is closely related to its problem data. Through comparison of the data of two problems, the similarity level of their corresponding search spaces can be identified. If one problem has a search space similar enough to that of another, the solution derived for the first problem can possibly be used as a basis for uncovering useful knowledge or a meme, akin to being a "shortcut" to the optimum solution for the second. The meme has a high level of possibility to enhance the performance of the search for the second problem. The similarity level between search spaces of different problems is an intuitive criterion for the selection of such memes. The method of selecting and implementing existing solutions of solved problems as memes embedded in the initial solutions to help finding the optimum for an unsolved problem is therefore desirable.

1.3 Objectives

Based on the motivations mentioned in Section 1.2, the target is to develop a robust and flexible search methodology capable of capitalizing on multiple memes so as to perform consistently on various types of combinatorial optimization problems. The following details present the steps of how the objective is achieved.

Firstly, a collaborative memes framework (CMF) that incorporates and manages multiple memes is proposed. This framework allocates independent population pools for different memes to ensure that their unique contributions are well preserved. A communication protocol is established between each pair of memes to guarantee the collaborative coupling between memes and keep a healthy balance between population diversification and improvement of solution quality is maintained.

Secondly, a neural meta-memes framework (NMMF) is proposed. Based on the fact that different types of problems favor different meta-memes to produce the best results, NMMF is designed with the capability of generating different optimized meta-meme methods for various problems. This framework achieves the above target by utilizing the experience acquired from earlier problem-solving sessions to direct the construction of appropriate meta-meme approaches for previously unsolved problems.

Thirdly, a method of utilizing existing solutions as memes to manipulate the initial search area for a problem is proposed. This method utilizes existing solutions of solved

problems as memes in the form of solution fragments of the initial solutions of a previously unsolved problem. The target is to increase the search efficacy and reduce the computational cost.

1.4 Contributions

Empirical studies show that the performance of the two proposed frameworks and the method of utilizing solutions as memes in the form of solution fragments met the expectations. The proposed collaborative memes framework (CMF) improves the solutions quality significantly for a group of large-scale QAP benchmarks as compared to the independent memes which CMF consists of. CMF serves as a platform allowing different memes to carry out search with their own unique procedures on their private search spaces. Meanwhile, the capability of communication ensures collaboration between its constituent memes and therefore maintains the robustness brought about by hybridization. CMF is a successful trial of constructing a collaborative approach for incorporating multiple memes with clearly defined communication protocol, targeting at balancing the population diversification and achieving good solution quality for combinatorial optimization problems.

According to the "*no free lunch theorem*", no single approach can perform consistently well on all types of problems. Here the neural meta-memes framework (NMMF) is

designed to incorporate different memes and manage them differently so as to produce variants of optimized search mechanisms for different problems. Different from the CMF, the NMMF is a problem-adaptive approach and therefore it has much higher flexibility. The main novel features of NMMF can be summarized as follow:

1) The experience in solving one combinatorial optimization problem is used to direct the construction of optimized meta-meme method for another problem through the collaborative coupling between different components within NMMF.

2) NMMF provides a scheme capable of reacting or handling differently the different types of combinatorial optimization problems in order to achieve consistently good performance.

During testing, NMMF generated appropriate control mechanisms for groups of small-scale combinatorial optimization problems through a training data generation scheme. These generated mechanisms represent the experience acquired from solving the corresponding combinatorial optimization problems. The mechanism generation system consists of a neural network which is trained with dataset containing the generated mechanisms by the training data generation module and their associated problem data. The NMMF with trained neural network was tested with a class of difficult combinatorial optimization problems, the QAPs. NMMF successfully generates appropriate meta-meme approach for every problem under test. The experimental results are compared with recently published results produced by another powerful method on the same problem instances. The NMMF outperforms that method, with better success rates and solutions

quality for most of the instances. The comparison attests to the robustness of NMMF as an efficient and effective method for combinatorial optimization problems. Additional tests were carried out for the same suite of problems with meta-meme approaches different from those generated by NMMF. The results are worse off compared to the generated approaches. Extensive testing on a group of new QAP benchmarks designed to stretch the capability of stochastic methods further demonstrates the robustness of NMMF. The memes implemented within NMMF can operate independently as black boxes; such configuration leads to an open and user-friendly framework with flexibility to incorporate new and novel methods.

The method of utilizing existing solutions as memes was tested with groups of QAP benchmarks. Qualified solution fragments providers were found for a number of instances. These instances were tested with the selected memes embedded in their associated initial solution permutations. The experimental results were compared with that generated with randomly constructed initial populations. The empirical study shows that the utilization of such memes effectively reduces the computational time cost for most of the cases. Extensive experiments with a lower similarity level criterion for selection of suitable meme providers were carried out on more instances. It shows that memes selected based on insignificant similarity level are not effective in improving the search efficiency.

1.5 Organization of the Thesis

This thesis is organized as follows.

This chapter gives an introduction to the basic concepts in combinatorial optimization problems, metaheuristic methods or memes. The motivations, objectives and contributions of the work in this thesis are also discussed.

In Chapter 2, literature review on combinatorial optimization problems including the quadratic assignment problem (QAP) and several popular search methods including the four adopted in the work are presented with details.

Chapter 3 presents the proposed collaborative memes framework.

In Chapter 4, the neural meta-memes framework is discussed with implementation and simulation details.

Chapter 5 focuses on the method of utilizing existing solutions as memes in the form of solution fragments in solving quadratic assignment problems.

In Chapter 6, conclusions on the work including limitations and future research directions are presented.

Chapter 2 Combinatorial Optimization and Metaheuristics

2.1 Complexity of Combinatorial Optimization Problems

The objective of a decision problem is to validate the truth or falsity of a statement. For a combinatorial optimization problem, the objective is to find the solution with maximal or minimal cost value from a finite number of solutions, depending on whether the problem is a maximization or minimization problem.

In computational complexity theory, decision or optimization problems which can be solved in polynomial time $O(n^c)$ (c is some constant, n is the input size) are denoted by *P*, while *NP* is the class of problems which an input or a solution to the problems can be verified or evaluated in polynomial time, the term "*NP*" stands for "non-deterministic polynomial time". Problems in *P* belong to *NP*, that is $P \subseteq NP$. A problem *p* from *NP* is in the *NP-complete* class if and only if all the *NP* problems can be reduced to it in polynomial time. The complexity class *NP* includes most of the combinatorial

optimization problems.

The hardest within the class *NP* is the *NP-complete* problems which were introduced in (Cook 1971). Any solution to the *NP-complete* problems can be verified in polynomial time, but there is no efficient method to locate a solution or to solve the *NP-complete* problems. As the size of the *NP-complete* problem increases, the number of solutions and the time required to solve it increases exponentially. Even with current computing technology, millions or billions years of computational time cost are expected to solve only moderately large-scale *NP-complete* problems by existing deterministic search methods. Although theoretically there is still the possibility that the two classes *P* and *NP-complete* are the same, that is *P=NP*, this issue remaining as one of the greatest open questions in computer science. It is already widely recognized that a method which is capable of solving *NP-complete* problems in reasonable polynomial time is unlikely to exist based on overwhelming empirical evidence.

The other class of problems is the *NP-hard* problems refer to the problems which are at least as hard as the hardest problems in *NP*. *NP-hard* problems do not have to be decision problems. Most of the real-world optimization problems are *NP-hard* and there are no existing approaches to solve them in polynomial time. Examples of *NP-hard* problems include bin packing problem, vehicle routing problem, N-queens problem, travelling salesman problem, knapsack problem, etc. The relationships between *P*, *NP*, *NP-complete* and *NP-hard* are depicted in Figure 2.1 for the two situations: $P \neq NP$ and $P=NP$.

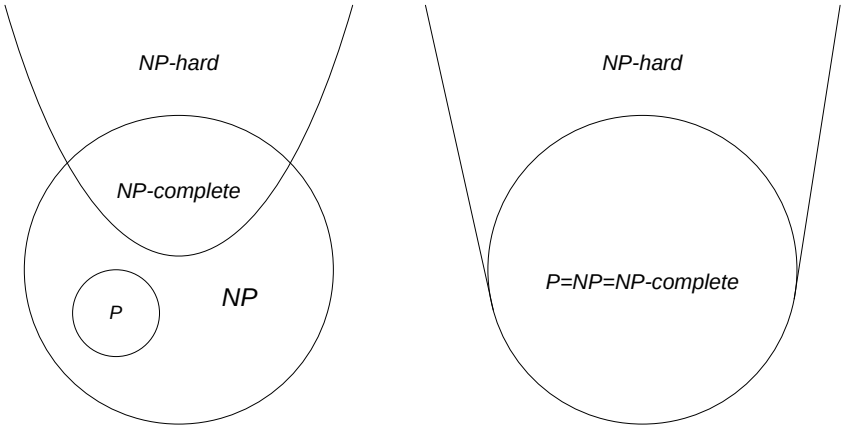


Figure 2.1 Euler diagram for *P*, *NP*, *NP-complete* and *NP-hard* problems: (a) $P \neq NP$; (b) $P = NP$

In most of the cases, the complexity of combinatorial optimization problems is increasing exponentially with the size of the problem. For example, the N-queens problem is about locating n (n is the length of the problem) queens on an $n \times n$ chessboard in mutually non-attacking positions. A feasible solution for the problem is a permutation of size n. There are totally $n!$ permutations for a problem of size n; in other words, the size of the search space is $n!$. Another example is the travelling salesman problem (TSP) which is perhaps the most popular combinatorial optimization problem. A TSP of size n can be formulated as follows: given n cities and a distance matrix D of size $n \times n$; d_{ij} is the distance between city i and city j; the objective is to find a tour which visits each city exactly once and minimizes the total distance. Similarly, the total number of solutions is $n!$. As the problem size increases, the search space or the number of possible solutions for an N-queens problem or a TSP increases exponentially. Generally, problems of size larger than 20 are considered to be not solvable through deterministic methods. As such,

stochastic methods are more appropriate to derive good solutions in reasonable time for problems of greater dimension and complexity.

2.2 Combinatorial Optimization

In general, optimization refers to a process of finding "the best" from a set of possibilities. Numerous real-world optimization problems are formulated in different areas, such as engineering, science and economics. Most of them are complex and difficult to solve. Among the different optimization problems, the combinatorial optimization problems are the class of problems which deals with discrete problems. The objective of combinatorial optimization is to find the optimum from a large but finite number of solutions. In (Lawler 1976), the combinatorial optimization is defined as "the mathematical study of finding an optimal arrangement, grouping, ordering, or selection of discrete objects usually finite in numbers."

A typical combinatorial optimization problem consists of a set of instances $\Omega=\{I_1, I_2, I_3, \ldots, I_n\}$ assuming the number of instances is a positive integer n; a search or solution space S_i is associated with each instance I_i which contains a set of solutions $\{s_1, s_2, s_3, \ldots, s_m\}$ with the assumption that the number of solutions for instance I_i is a positive integer m. An object function C for the problem evaluates and assigns solution cost value for every

solution in all the solutions sets. So the objective of the combinatorial optimization problem is to maximize or minimize the following depending on whether the problem is a maximization or minimization problem:

$$Maximize\ (or\ minimize)\ C(s_j)\ subject\ to\ s_j \in S_i \subseteq \Omega$$

where $i=\{1,2,3,\ldots,n\}$, $j=\{1,2,3,\ldots,m\}$, n is the number of instances of the problem, and m is the number of solutions in solutions set S_i associated with instance I_i.

Many practical applications of combinatorial optimization problems are *NP-hard*; there is no polynomial method that can solve these problems currently and the possibility of such a method not existing is high. Therefore the study and development of stochastic methods which can achieve good approximate solutions for these complex problems are important. In this chapter, the metaheuristics, possibly the most popular stochastic methods in recent decades, are extensively introduced. In the following section, details of the quadratic assignment problem (QAP) which is arguably one of the hardest combinatorial optimization problems are presented. It is extremely difficult to solve some of the existing QAP benchmarks by both exact and stochastic methods. This makes the QAP a good problem test set to validate the robustness and effectiveness of optimization methods.

2.3 Quadratic Assignment Problems

The quadratic assignment problem (QAP) was formulated by Koopmans and Beckmann (1957) to solve the cost minimizing problem of assigning a number of plants to the same number of locations. Many practical applications can be formulated as QAPs, such as backboard wiring problem (Steinberg 1961), electronic components layout problem (Or and Atik 2000, Rabak and Sichman 2003, Miranda et al. 2005, Duman and Or 2007), hospital layout problem (Elshafei 1977), campus planning (Dickey and Hopkins 1972), facility layout problem (Mavridou and Pardalos 1997), machine scheduling (Back et al. 1997), ranking of archaeological data (Krarup and Pruzan 1978) and so on.

The QAPs are among the hardest combinatorial optimization problems (Sahni and Gonzalez 1976). A QAP of size n may be stated as assigning n facilities to n locations while knowing the distance between every two locations and the flow between every two facilities. The objective evaluation function for a QAP can be formulated as the following equation:

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} \tag{2.1}$$

In Eq. (2.1), matrix A is the distance matrix, a_{ij} represents the distance between location i and location j; matrix B is the flow matrix, $b_{\pi(i)\pi(j)}$ represents the flow between facility $\pi(i)$ and facility $\pi(j)$. Here π is a permutation of the set $M = \{1,2,3,\ldots,n\}$ and $\pi(i)$ denotes that facility $\pi(i)$ is assigned to location i. The permutation which produces the minimum cost

value with Eq. (2.1) is the optimum solution for a QAP. The total number of solution permutations for a QAP of size n is $n!$.

Different deterministic methods are introduced to solve QAPs, such as branch and bound, cutting planes, etc. The branch and bound methods are the most well studied exact methods. The lower bounds are the crucial components of the branch and bound methods. A lower bound can also be used to evaluate the quality of the solutions obtained by heuristic methods. Gilmore-Lawler bound (Gilmore 1962, Lawler 1963) is one of the most well-known lower bounds. The number of feasible solutions of the QAP increases exponentially with its size. In practice, QAPs with size around 30 are the largest instances solvable by exact methods except for some specially structured QAPs (Burkard et al. 1997).

In recent years, researchers have resorted to approximate search approaches including metaheuristic methods to manage the time-complexity of finding good solutions to the QAPs. In particular, the last 3 decades have seen the proliferation of metaheuristic methods such as greedy randomized adaptive search procedure (GRASP) (Li et al. 1994), genetic algorithms (Tate and Smith 1995), hybrid genetic algorithms (or memetic algorithms) (Fleurent and Ferland 1994, Merz and Freisleben 1999, Lim et al. 2000, Merz and Freisleben 2000, Lim et al. 2002, Xu et al. 2003, Drezner 2003, Drezner 2005a, Tang et al. 2006, Tang et al. 2007), evolution strategies (Nissen 1994), particle swarm optimization (Gong and Tuson 2008), ant colony optimization (Stützle 1997, Taillard and Gambardella 1997a, Stützle and Dorigo 1999, Maniezzo and Colorni 1999, Gambardella

et al. 1999a, Talbi et al. 2001, Middendorf et al. 2002, Xu et al. 2006), simulated annealing (Connolly 1990, Laursen 1993, Misevicius 2003), tabu search (Skorin-Kapov 1990, Taillard 1991, Skorin-Kapov 1994, Battiti and Tecchiolli 1994, Taillard 1995, Bachelet et al. 1996, Misevicius 2005, Drezner 2005b), neural networks (Chakrapani and Skorin-Kapov 1992, Ishii and Sato 1998, Azim 2006), scatter search (Cung et al. 1997), iterated local search (Stützle 2006), variable neighborhood search (Taillard and Gambardella 1997b), etc. The various methods for solving QAPs are summarized in (Loiola et al. 2007).

Due to the high difficulty level and the availability of a wide range of diverse problem instances, the QAP benchmarks are chosen to test the performance of the work in this thesis for combinatorial optimization problems. The benchmarks used in the experiments can be classified into groups based on their attributes. The number tagged to the name of a benchmark indicates the size of that problem. The whole suite of QAP benchmarks can be classified into several categories as described in (Lim et al. 2002).

Real-life and classical problems - This category contains the problems which stem from real-world applications of the QAPs. There are several groups in this category. The *Burxx* problems by Burkard and Offermann were designed to find the best typewriter keyboards for various languages. *Els19* was designed for solving the hospital layout problem; the objective is to minimize the total walking distance by people in a hospital. The Steinberg's problems, abbreviated as *stexx*, are instances modeling the computer components layout problems. The *escxx* problems are for testing of self-testable units in

computer science. *Kraxx* are benchmarks containing data used for designing real-world building layout problems. *Nugxx* type benchmarks presented by Nugent et al. (1968) are problems of facility layout design. The distance matrices of *nugxx* problems contain Manhattan distances of rectangular grids. Other types like *hadxx*, *scrxx*, *thoxx* and *wilxx* were also derived from facility layout design problems. In general, the size of the real-world problems is relatively small and most of them are relatively simple. The values in the flow and distance matrices are not uniformly generated. Compared to other groups, there are more zero elements exist in the flow matrices in this category.

Generated problems - The *taixxb* series benchmarks were generated by Taillard with similar distribution as in real-life problems (Taillard 1991, Taillard 1995). This group is also known as the real-life like benchmarks. These problems are asymmetric and randomly generated with problem size up to 150. *Taixxa* benchmarks also by Taillard (Taillard 1991, Taillard 1995) were generated randomly with a uniform distribution. *Taixxa* problems are unstructured. Most of the iterative heuristic methods can easily achieve solutions around 1% above the best known solutions, but solving this type of problems up to their optima is extremely difficult. The *lipaxx* problems were generated by Li and Pardalos (1992) with known optimum solutions by certain algorithms. The matrices of this type of benchmarks are asymmetric. The *skoxx* group proposed by Skorin-Kapov (1990) is the type of benchmarks with rectangular distances and pseudo-random entries in the flow matrices. The problem size of this type is from 42 to 100. The *rouxx* benchmarks were generated with entries chosen from 0 to 100 (Burkard

et al. 1997).

2.4 Metaheuristics

In (Polya 1945), the heuristic concept in optimization was introduced. The heuristic approaches refer to the approximate methods based on random search which find solutions among all the feasible solutions for a problem. Different from the exact methods which are usually based on some enumeration techniques, heuristic methods do not guarantee that the optimum solution will be found. The term "heuristic" comes from "heuriskein" which means "find" or "discover". Although exact methods guarantee discovering the optimum, but in reality the computational time cost turns out to be unacceptable for most of the large-scale combinatorial optimization problems. Usually heuristic methods find solutions close to the optimum in short computational time, sometimes uncovering the optimum solution. This provides the motivation for developing robust and efficient heuristic methods.

Metaheuristics are types of search methods which utilize basic heuristic methods and at the same time carry out the search in a more global point of view. The term metaheuristic was introduced by Glover (1986), the prefix "meta" means "beyond" or "upper level". For traditional heuristic methods, the concepts and the search procedures are simple and the efficiency of the methods is relatively low. In metaheuristic methods, multiple search

procedures of two or more basic heuristic methods are combined to tackle the two contradictory criteria, the exploration and the exploitation. The exploration or diversification refers to exploring the search space while the exploitation or intensification is to exploit a single solution to find even better solutions. The exploration helps to jump from one search area to another in order to cover as much as possible the expanse of the whole search space. The exploitation is capable of locating the local optimum in one search area. The metaheuristic methods manage multiple basic heuristic methods with strategies from the upper level to balance the exploration and exploitation to achieve the best complementation between the two. Heuristic methods normally are biased on one of the two aspects. For example, the genetic algorithm explores the search space through crossover between individuals while the pair-wise local search method exploits an individual by searching within its neighborhood. In general metaheuristic methods are excellent in solving combinatorial optimization problems, especially for large-scale problems. Various metaheuristic methods can be classified into different classes, for example, based on the number of individual solutions being searched, metaheuristics can be divided into single solution based and population based methods. Single solution based methods include local search, simulated annealing, tabu search, guided local search, iterated local search, etc. The examples of population based methods are genetic algorithm, ant colony optimization, particle swarm optimization. In recent years with the fast improvement in the computer power and memory size, most of the metaheuristic methods researched are hybrid of more than one single metaheuristic method and a population of individuals is necessary for their search procedures to work.

In the work presented in this thesis, all the standard metaheuristic methods are treated equally as knowledge systems or memes including fixed search procedures to be applied to population of individuals. In the following sections of this chapter, several popular metaheuristic methods including those adopted in the work here are elaborated with details. Glover and Kochenberger (2003) elaborate various metaheuristic methods with implementation details and applications. In (Talbi 2009), a comprehensive view is given on existing metaheuristic methods by dividing them into categories.

2.4.1 Genetic Algorithm

Evolutionary algorithms refer to the class of search methods which were inspired by Darwin's theory of evolution. Genetic algorithm (GA), introduced by Holland (Holland 1962, Holland 1975), is one of the most well-known evolutionary algorithms. Other evolutionary algorithms (EAs) include evolution strategies (Rechenberg 1973), evolutionary programming (Fogel 1962, Fogel et al. 1966) and genetic programming (Koza 1992). Genetic algorithm has been popular for the last two decades and reasonably successful in solving optimization problems with various applications in a wide range of areas, such as TSP (Freisleben and Merz 1996), machine learning (Goldberg 1989), bioinformatics (Van Batenburg et al. 1995), etc. Since its introduction, studies on different aspects of genetic algorithm were carried out by researchers. More details about genetic algorithm and other EAs can be found in (Back et al. 1997, Yao 1999, Glover and Kochenberger 2003, Talbi 2009).

The search process of a GA starts with a randomly generated population of individuals,

which are some kind of representations to the solutions. Examples include permutation representation and binary representation. A random initialized population of individuals is considered to maximize the diversity of the initial population when there is no prior knowledge known for the optimum solution of an optimization problem. The individuals are also known as chromosomes with respect to the genetic operators. Each individual has a fitness value derived by evaluating the individual with an objective function. Genetic algorithms emulate natural evolution to solve optimization problems. Hence, the working population undergoes operations such as parent selection, crossover, mutation, and so on. The *parent selection* step picks individuals as parents. The selected parents undergo *crossover* (or *recombination*) to generate offspring. The *mutation* step modifies an individual's gene based on a preset mutation rate. In the final stage of every generation, the offspring are merged into the population based on some replacement strategies. The above steps are repeated until a termination criterion is reached. The stopping criteria can be:

1. The number of generations reaches a preset limit.

2. Allocated computational time budget is reached.

3. The diversity of the population reaches a certain level.

4. A solution satisfying the requirement is found.

The *parent selection* operator is important for keeping the diversity and at the same time allowing the best individuals to survive and improve. The selection is carried out

according to some selection strategies such as roulette wheel selection, tournament selection, stochastic universal sampling, and so on. The fitness values of the individuals are the basic reference parameters in most of the strategies, for example, in roulette wheel selection, the individuals with larger fitness values are allocated higher chances to be selected as parents for generation of offspring. With roulette wheel selection, each individual is given a probability to be selected exactly proportionate to its fitness. Outstanding candidate solutions biased in the beginning of every generation may cause the fast reduction of diversity and hence the early premature convergence of the population. Meanwhile, the stochastic universal sampling (Baker 1987) is an extension of the roulette wheel selection. With stochastic universal sampling, a random number is used to select several individuals on the evenly spaced intervals on the total fitness space. The tournament selection involves randomly choosing a fixed number of individuals each time, the best is eventually selected. This process repeats until enough individuals are selected as parents. In (Hancock 1994), different selection methods are tested and compared with each other. The stochastic universal sampling method outperforms the roulette wheel selection. This further enhances the notion that population diversity is crucial for the performance of a GA.

With *crossover*, or *recombination* operator, two or more parents exchange parts of their genes and produce offspring individuals. There are also different ways of performing crossover, such as one-point crossover, two-point crossover and uniform crossover. All the genetic information of the offspring is inherited from the parents. Crossover has an

effect of confining population diversity. The *mutation* operator which is complementary to the *crossover* operator randomly modifies a small part of an individual, introducing more diversity to the population. A replacement strategy is used to determine which individuals survive into the next generation from the original population and the newly generated offspring individuals. As proposed by J. Holland, the parent population is replaced by the offspring population systematically. The steady-state replacement on the contrary only allows one offspring generated and replaced into the working population in every generation. A good description about this type of GAs can be found in (Davis 1991). Besides these two extreme strategies there are other schemes proposed. The *elitism* and *population overlaps* strategy proposed by De Jong (1975) is a straightforward method which preserves the best one and replaces the rest with offspring individuals. A good replacement scheme should be able to keep the diversity level and avoid premature convergence of a population.

For a typical GA, population size (number of individuals in one population), crossover rate (proportion of individuals undergo crossover with reference to the population size), mutation rate (probability for an individual undergoes mutation) and mutation range (proportion of elements undergo mutation with reference to the total number of elements on an individual) are important control parameters. In general, a small size population converges faster but the solution quality is worse compared to a large size population. But if the size is too large, computational time cost will not be acceptable for the population to converge to some good quality solutions. The population size should be

chosen so as to achieve a healthy balance between the convergence speed and the solution quality. In practice, a number chosen from 20 to 100 is deemed to be an appropriate population size. A large mutation rate will destroy the population and lead to a random search. Typically, a small mutation rate is desirable, for example, a number in [0.001, 0.01].

Pure genetic operators only provide exploration of the search space; the solutions discovered especially for large-scale combinatorial optimization problems are generally not satisfactory. In practice, pure genetic approach normally collaborates with other heuristic methods to provide efficient ways of refining individual solutions, generally referred to as hybrid genetic algorithms (HGAs), or memetic algorithms (MAs). The exploratory nature of genetic operators plus the exploitive nature of local search methods result in more robust and efficient methods on solving different types of optimization problems. For a traditional MA, the local refinement is modeled as an individual's self learning process. The self learning process is metaphorically considered as a meme which by its original definition is a unit of cultural evolution and hence the name "memetic" is given. More details about the development of MAs are discussed in Section 2.5. The pseudo-code for the GA is given in Figure 2.2. The pseudo-code shown assumes that GA is applied on a minimization problem.

Procedure::Genetic Algorithm (working population with size *n*)
BEGIN
Initialize mutation rate m_{rate}, mutation range m_{range}, crossover rate c_{rate}, number of parents selected each generation p_{parent}
Initialize the best solution found $s_{bestFound}$ with the best solution in the working population
Do
Begin
Initialize number of selected $n_{parent} := 0$
Initialize number of offspring generated $n_{offspring} := 0$
Initialize an offspring population with size of $p_{parents}$
While ($n_{offspring} < p_{parent}$)
Randomly generate a number r_1 between (0, 1)
If $r_1 < c_{rate}$, then
Select two individuals (s_1, s_2) from working population randomly which were not selected before in this generation
Produce two new individuals (s_3, s_4) by applying two point crossover on (s_1, s_2)
Randomly generate a number r_2 between (0, 1)
If $r_2 < m_{rate}$, then
Generate $max(2, m_{range} * n)$ random positions and swap the elements on these positions on s_3
$s'_3 := Mutate(s_3)$;
Generate $max(2, M_{range} * n)$ random positions and swap the elements on these positions on s_4
$s'_4 := Mutate(s_4)$;
Else
$s'_3 := s_3$;
$s'_4 := s_4$;
End If
Increase n_{parent} by 2
If s'_3 and s'_4 are different from s_3 and s_4
Add individuals S'_3, S'_4 into the offspring population
Increase $n_{offspring}$ by 2
End If
End If
End While
Replace the worst $n_{offspring}$ individuals in current working population with $n_{offspring}$ offspring individuals from offspring population
Update the best solution found $s_{bestInPop}$ with the best solution in current working population
If ($s_{bestInPop} < s_{bestFound}$), then
$s_{bestFound} := s_{bestInPop}$
End If
End While
End

Figure 2.2 Pseudo-code of a genetic algorithm

2.4.2 Local Search

Local search is probably the simplest and oldest metaheuristic method for combinatorial optimization problems (Flood 1956, Croes 1958). A typical local search starts with an initial candidate solution. At every generation of the search, a neighbor with better

solution quality replaces the current solution. A neighbor for an individual solution is obtained by a one step modification or a *move* to that solution. For example, for a solution represented by a binary string, a *move* refers to a flip to one of its bits; for a permutation based solution, a *move* is the swap between two of the elements on the permutation. The search continues until there is no neighbor better than the solution itself.

The selection of a neighbor in every generation is based on some selection strategies. The *first improvement* strategy evaluates a solution's neighbors in a given order until a neighbor with better solution quality is found. That neighbor is the search result for the present generation and replaces the solution. The *best improvement* approach examines the neighbors of a solution exhaustively at every generation and the best neighbor is selected. Other approaches such as random selection are also proposed. The *first improvement* method requires much less computational time compared to the *best improvement* but the solution quality is normally worse off.

```
Generate an initial individual solution s;
repeat
    generate neighbor s' for the neighbors set N(s) of s ;
    if s' is better than s
        s := s';
until all s' ∈ N(s) are not better than s
```

Figure 2.3 Pseudo-code of a local search

In most cases, the search results of a local search are only local optima. For the

optimization problems with too many local optima in their search spaces, local search may not be an effective method to find the global optimum solution. Many heuristic methods have been proposed to help the local search to work in a more global way by incorporating capabilities to escape from the local optima. Such methods can be classified as extensions of local search, such as guided local search, iterated local search, tabu search, simulated annealing, etc.

Additionally, since a local search method usually carries out small adjustments to a solution, there are always efficient "shortcuts" available for the calculation of the solution cost difference. For example, a permutation based solution π for a QAP undergoes a *k*-gene exchange local search with which *k* alleles on the solution permutation are swapped. Given the problem size *n* and the condition $2 \leq k \leq n$, let the *k* alleles undergoing exchange be denoted as $i_1, i_2, \ldots, i_k$, and the set $M=\{1,2,\ldots,n\}$ be divided into $M_1=\{i_1,i_2,\ldots,i_k\}$ and $M_2=M-M_1$. Let π^* denotes the solution after being applied the *k*-gene exchange and δ is the cost difference between π^* and π, i.e. $\delta=C(\pi^*)-C(\pi)$. The values of $C(\pi^*)$ and $C(\pi)$ can be calculated directly from Eq. (2.1), but this introduces significant computational cost. A more efficient method is to calculate only the items involved with those exchanged alleles. This method is elaborated by discussing the details of the *k*-gene exchange. The *k*-gene exchange procedures are as follow:

$$\pi^*(l) = \pi(l), \quad for\ all\ l \in M_2$$

$$\pi^*(i_1) = \pi(i_2),$$

$$\pi^*(i_2) = \pi(i_3),$$

$$\vdots$$

$$\pi^*(i_k) = \pi(i_1). \tag{2.2}$$

With the definition $\pi(i_{k+1}) = \pi(i_1)$, the general form of Eq. (2.2) can be written as:

$$\pi^*(l) = \pi(l), \quad for\ all\ l \in M_2$$

$$\pi^*(i_r) = \pi(i_{r+1}), \quad for\ all\ i_r \in M_1 \tag{2.3}$$

The objective function for the QAP as computed by Eq. (2.1) can be rewritten as:

$$C(\pi) = \sum_{l\in M_2}\sum_{t\in M_2} a_{lt}\, b_{\pi(l)\pi(t)} + \sum_{l\in M_1}\sum_{t\in M_2} a_{lt}\, b_{\pi(l)\pi(t)}$$

$$+ \sum_{l\in M_2}\sum_{t\in M_1} a_{lt}\, b_{\pi(l)\pi(t)} + \sum_{l\in M_1}\sum_{t\in M_1} a_{lt}\, b_{\pi(l)\pi(t)} \tag{2.4}$$

Based on Eq. (2.4), $C(\pi^*)$ and $C(\pi)$ are only different in the last three items in Eq. (2.4), the cost difference δ can be calculated as follow:

$$\delta = C(\pi^*) - C(\pi)$$

$$= \sum_{t\in M_2}\sum_{r=1}^{k}[a_{i_r t}\left(b_{\pi(i_{r+1})\pi(t)} - b_{\pi(i_r)\pi(t)}\right) + a_{t i_r}\left(b_{\pi(t)\pi(i_{r+1})} - b_{\pi(t)\pi(i_r)}\right)]$$

$$+ \sum_{r=1}^{k}\sum_{s=1}^{k}[a_{i_r i_s}\left(b_{\pi(i_{r+1})\pi(i_{s+1})} - b_{\pi(i_r)\pi(i_s)}\right)] \tag{2.5}$$

With Eq. (2.5), the computational complexity of the *k*-gene exchange is improved from $O(n^2)$ with Eq. (2.1) to $O(kn)$.

Furthermore, for the most popular 2-gene exchange local search method which is the

simplest case of the *k*-gene exchange, let the two elements p and q located at $\pi(p)$ and $\pi(q)$ being swapped. The resulting cost difference is:

$$\delta(\pi, p, q) = C(\pi^*) - C(\pi)$$

$$= \sum_{i=1\&i\neq p,q}^{n} [(a_{ip} - a_{iq})(b_{\pi(i)\pi(q)} - b_{\pi(i)\pi(p)}) + (a_{pi} - a_{qi})(b_{\pi(q)\pi(i)} - b_{\pi(p)\pi(i)})]$$

$$+(a_{pp} - a_{qq})(b_{\pi(q)\pi(q)} - b_{\pi(p)\pi(p)}) + (a_{pq} - a_{qp})(b_{\pi(q)\pi(p)} - b_{\pi(p)\pi(q)}) \quad (2.6)$$

With Eq. (2.6), the computational complexity level is in $O(n)$. Now consider the case that elements u and v are swapped on a solution permutation π with $\{p, q\} \cap \{u, v\} \neq \emptyset$, and the cost difference caused by swapping elements p, q on π is known. Now let θ denotes the solution permutation achieved by swapping elements u, v on π. A more efficient way is available for calculating the solution cost difference caused by swapping elements p, q on θ with the existing cost difference by swapping p, q on π. This "shortcut" was introduced by Taillard with robust tabu search introduced in (Taillard 1991). The cost difference by swapping p, q on θ can be calculated by examining only those elements in Eq. (2.6) which are affected by swapping (u, v):

$$\delta(\theta, p, q) = \delta(\pi, p, q) +$$

$$[(a_{up} - a_{uq})(b_{\theta(u)\theta(q)} - b_{\theta(u)\theta(p)}) + (a_{pu} - a_{qu})(b_{\theta(q)\theta(u)} - b_{\theta(p)\theta(u)})$$

$$+(a_{vp} - a_{vq})(b_{\theta(v)\theta(q)} - b_{\theta(v)\theta(p)}) + (a_{pv} - a_{qv})(b_{\theta(q)\theta(v)} - b_{\theta(p)\theta(v)})]$$

$$-[(a_{up} - a_{uq})(b_{\theta(v)\theta(q)} - b_{\theta(v)\theta(p)}) + (a_{pu} - a_{qu})(b_{\theta(q)\theta(v)} - b_{\theta(p)\theta(v)})$$

$$+\left(a_{vp}-a_{vq}\right)\left(b_{\theta(u)\theta(q)}-b_{\theta(u)\theta(p)}\right)+\left(a_{pv}-a_{qv}\right)\left(b_{\theta(q)\theta(u)}-b_{\theta(p)\theta(u)}\right)] \quad (2.7)$$

After rearranging, Eq. (2.7) can be written as:

$$\delta(\theta,p,q)=\delta(\pi,p,q)+$$

$$\left(a_{up}-a_{uq}+a_{vq}-a_{vp}\right)\left(b_{\theta(u)\theta(q)}-b_{\theta(u)\theta(p)}+b_{\theta(v)\theta(p)}-b_{\theta(v)\theta(q)}\right)+$$

$$\left(a_{pu}-a_{qu}+a_{qv}-a_{pv}\right)\left(b_{\theta(q)\theta(u)}-b_{\theta(p)\theta(u)}+b_{\theta(p)\theta(v)}-b_{\theta(q)\theta(v)}\right) \quad (2.8)$$

With Eq. (2.8), the solution cost difference caused by a 2-gene exchange can be evaluated with complexity level of $O(1)$. The number of 2-gene exchange neighbors for a solution is $\frac{n(n-1)}{2}$, within all these neighbors, only ($2n$-3) of them are not mutually exclusive to a specific pair swap. With Eqns. (2.6) and (2.8), all neighbors of a solution can be evaluated in $O(n^2)$ time.

2.4.3 Guided Local Search

The guided local search (GLS) was introduced by Voudouris and Tsang (1999). To apply GLS, a set of features are defined with the solutions of a problem, each feature is associated with a cost and a penalty. The penalty values which are initialized to zero at the beginning are incorporated proportionally into the objective function. GLS starts the search with some local search method on a candidate solution. When the local search is trapped in a local optimum, certain features closely related to that local optimum are penalized through the increments on their penalty values. After being penalized, this local optimum is no longer "optimum" with the evaluation of the updated objective function

and the local search escapes the trap. Through the penalty scheme, GLS achieves diversification and carries out search on various regions with local optima on the search space of an optimization problem. The successful applications of GLS include TSP (Holstein and Moscato 1999), vehicle routing problem (Kilby et al. 1999, Kilby et al. 2000), and many more.

2.4.4 Iterated Local Search

Iterated local search (ILS), also known as chained local optimization, large-step Markov chains, is a metaheuristic method which is based on simple but powerful principle used in many heuristics such as the iterated Lin-Kernighan heuristic for the TSP (Johnson 1990). Starting with an initial solution (usually randomly generated), a local search procedure is applied on that solution. At every generation, a perturbation is carried out followed by the local search procedure on the solution. The search stops until a pre-specified criterion is satisfied. The perturbation method usually referring to a large number of random moves is responsible for redirecting the search to another area on the search space instead of remaining trapped in some local optimum. The strength of perturbation can be fixed or adaptive, depending on the problem and the search situation. In the simulations, the perturbation size is varied between two values k_{min} and k_{max}, this is similar to the strategy described in (Stützle 2006). At the beginning k is set to k_{min}, if after one generation of perturbation and local search there is no better solution found, increase k by 1. Otherwise, k is reset back to k_{min}. If k reaches k_{max}, it is also reset back to k_{min}. The local search method adopted can be a basic local search. The only task of the local search

is to find the local optimum in the current search area of the solution being examined.

```
Procedure:: Iterated Local Search (working population)
BEGIN
   Initialize k_max, k_min and worse solution acceptance criterion c := 0.0
   Initialize perturbation size k := k_min
   Initialize the best solution found s_bestFound with the best solution in the working population
   While (stopping conditions are not satisfied)
   Begin
      For each solution s_i in working population
         Set local optimum s_loc to the maximum integer
         Set counter j := 1
         While (s_i - s_loc) ≤ 0 or j is less than the number of neighbors of s_i
         Begin
            If neighbor s_j' is better than s_loc or (s_j' - s_loc) < 0, then
                  Update local optimum s_loc := s'
            Else
                  Increase j by 1
            End If
         End While
         If (s_i - s_loc) > 0, then
            Update the solution s_i := s_loc
         Else If (s_i - s_loc) ≤ 0, then
            Randomly generate a number r from (0, 1)
            If  r < c
                Update s_i := s_loc
            End If
         End If
      End For
      Update the best solution found s_bestInPop with the best solution in current working population
      If (s_bestInPop ≥ s_bestFound), then
         Initialize counter i := 1
         While i ≤ k
            Randomly generate 2 different positions in and swap the elements on the two positions
            Increase i by 1
         End While
         If k equals to k_max
            Set k := k_min
         Else
            Increase k by 1
         End If
      Else
         Set s_bestFound := s_bestInPop
         Set k := k_min
      End If
   End While
End
```

Figure 2.4 Pseudo-code of an iterated local search

An acceptance criterion is used to decide whether the uncovered local optimum at every

generation is qualified to replace the current solution while the local optimum is not better than the current solution. A good acceptance criterion should be able to keep healthy balancing between the intensification and the diversification of the search, such as the probabilistic acceptance criterion similar to the one adopted in simulated annealing. Iterated local search has been effectively applied in different areas, such as the QAP (Stützle 2006), job shop scheduling problem (Lourenco 1995), graph partitioning problem (Martin and Otto 1995, Martin and Otto 1996). More details about ILS can be found in (Lourenco et al. 2003). The pseudo-code for the ILS is given in Figure 2.4. The pseudo-code shown assumes that ILS is applied on a minimization problem.

2.4.5 Greedy Randomized Adaptive Search Procedures

Greedy randomized adaptive search procedures (GRASP) introduced in (Feo and Resende 1989) is a multi-start greedy metaheuristic for combinatorial optimization problems. The search of GRASP consists of two phases for each generation: construction and local search. During the construction stage, a feasible solution is built based on some greedy randomized heuristic method; then a local search is applied on that solution to find the local optimum. The greedy randomized construction method was first described in (Hart and Shogan 1987) as semi-greedy heuristic.

The GRASP can adopt any efficient method in local search phase such as tabu search, guided local search, etc. During the construction phase a feasible solution is constructed from scratch. At each generation of this phase, the elements which can be included into the uncompleted solution form a subset. The elements in the subset are evaluated with the

greedy randomized method and each of them is assigned an incremental cost value which indicates the cost change after including that element into the solution. A selection process selects a number of elements from the subset to form a restricted candidate list from which an element is randomly selected and included into the solution. The selection is based on some criterion, such as the value-based criterion. With this strategy, those elements with cost values less than a given threshold are selected. The fact that only the elements with the best incremental cost values are selected shows the greedy aspect of GRASP. The random selection of an element from the restricted candidate list shows the randomized or probabilistic aspect of GRASP. As a multi-start method, the performance of GRASP strongly depends on the construction phase which locates the search areas of the local search at each generation. Many applications of GRASP have been studied such as vehicle routing problem (Carreto and Baker 2002), MAX-SAT problem (Resende et al. 1997), QAP (Li et al. 1994).

2.4.6 Variable Neighborhood Search

Variable neighborhood search (VNS) is a rather recent metaheuristic method introduced by Hansen and Mladenovic (2001). The VNS is proposed based on the following experience on the search space of an optimization problem - one solution and its neighbor do not necessarily share the same local optimum and local optima have a tendency to be relatively close to each other. These facts indicate that a search through a solution's neighborhood chain may possibly uncover different local optima and even the global optimum solution. For a basic VNS, a set of predefined neighborhood structures

N_i ($i=1,2,\ldots i_{max}$) are provided. The search starts with an initial solution s, the first neighborhood structure N_1 of s is checked. The generation of the neighborhood structure of a solution in VNS normally refers to the *shaking* step. Let us denote the neighborhood structure N_1 of s as s'. A local search is applied to s' to generate new solution s''. If s'' is a better solution compared to s, s is replaced by s'', otherwise the original solution is kept. Then the same procedures carry on for the next neighborhood structure of s. The search stops until a stopping criterion is met. It is obvious that the balance between the diversification (by shaking) and the intensification (by local search) is crucial for the performance of VNS. Despite its short history, VNS has been applied in different areas, such as the vehicle routing problem (Gendreau et al. 2002), the minimum spanning tree problem (Ribeiro et al. 2002).

2.4.7 Simulated Annealing

Simulated annealing (SA) was first introduced by Kirkpatrick et al. (1983). It is a heuristic method which was based on the analogy between the solving of combinatorial optimization problems and the annealing process in metallurgy. The physical annealing process involves heating some solid (crystal) to a high temperature, then slowly cooling down to rearrange the molecules of the solid. The objective is to achieve thermal equilibrium for the crystal during the cooling process and the strong and stable crystal at the low ground state. The idea of simulated annealing originates from the Monte-Carlo method proposed by Metropolis et al. (1953) to simulate the energy changes in a physical system with different temperatures.

Procedure:: Simulated Annealing (working population)
BEGIN
Initialize temperature T, temperature annealing factor α, the search steps in one generation m
Initialize the best solution found $s_{bestFound}$ with the best solution in the working population
While (stopping conditions are not satisfied)
Begin
For each solution s_i in working population
Begin
Initialize count number $n_c := 0$
If $n_c < m$, then
Randomly generate a neighbor $s' := \text{swap}(s_i)$
If $(s' - s_i) < 0$, then
Update $s_i := s'$;
Else If $(s' - s_i) \geq 0$, then
Randomly generate a number r from (0, 1)
If $e^{\frac{-(s'-s_i)}{T_i}} > r$, then
Update $s_i := s'$;
End If
End If
Increase n_c by 1
End If
Update the current temperature $T = T * \alpha$
End For
Update the best solution found $s_{bestInPop}$ with the best solution in current working population
If $(s_{bestInPop} < s_{bestFound})$, then
$s_{bestFound} := s_{bestInPop}$
End If
End While
End

Figure 2.5 Pseudo-code of a simulated annealing

With simulated annealing, a solution to an optimization problem is modeled as an energy state of a system. In every generation, the individual solution undergoes the "annealing" process which involves a series of random moves. The maximum number of moves allowed in each generation m is a preset constant. Those moves which result in better solutions will be performed, otherwise a certain probability level to perform the move is accorded, and the probability of executing the move is derived based on Eq. (2.9).

$$P(\Delta E, T_i) = e^{\frac{-\Delta E}{T_i}} \tag{2.9}$$

where i is the generation number, ΔE is the energy level (solution cost) difference between the newly generated solution by that move and the current solution, T_i is the temperature value in the ith generation and P is the probability value. In the beginning of simulated annealing, the parameter T (as temperature) is initialized with a constant. After every generation T is recalculated using Eq. (2.10), where α is usually set to a value smaller than 1 but very close to 1.

$$T_{i+1} = T_i * \alpha; \qquad i = 1,2,3, \ldots \tag{2.10}$$

Simulated annealing provides the chance for the search to escape from local optima by allowing moves that worsen the solution with a certain probability. As the number of iterations increases, the temperature decreases and the probability to accept a worse neighbor decreases. In this way, the search focuses more on the exploitation of the search area to ensure a good solution can be uncovered. Due to its simplicity and flexibility, simulated annealing is applied to various optimization problems. In (Laursen 1993), simulated annealing on solving QAP is studied. An adaptive simulated annealing is proposed in (Chen and Luk 1999) to address optimization problems derived from signal processing applications. Other applications include school timetabling problem (Abramson et al. 1999), airline crew-pairing problem (Emden-Weinert and Proksch 1999), job shop scheduling problem (Van Laarhoven et al. 1992), etc. Hybrid methods between simulated annealing and other metaheuristic methods such as tabu search have been shown to provide better results than the independent methods, i.e. the non-hybrids (Osman 1993). Simulated annealing has also been successfully applied to solve

continuous optimization problems (Locatelli 2000). The pseudo-code for the SA is given in Figure 2.5. The pseudo-code shown assumes that SA is applied on a minimization problem.

2.4.8 Tabu Search

Tabu search (TS) was developed by Glover in (Glover 1989, Glover 1990). It is one of the most widespread metaheuristic techniques. The key feature of tabu search is the utilization of memory structures to store the information of the search procedures which helps to control the balance between exploration and exploitation.

A short term memory, or tabu list, is used within tabu search to record the previous moves of a candidate solution for an optimization problem. The moves recorded on the tabu list are forbidden to be taken for a given number of iterations, named the *tabu tenure*. Tabu tenure can be a variable for different search period. Tabu list helps to increase the efficiency of the search by avoiding repeated and unproductive search steps. However if a tabued move is capable of leading to a solution with quality good enough to satisfy an aspiration criterion, that move will be taken. One aspiration criterion adopted in many applications is that a tabued move generates a solution which is better than the best solution found so far. At every generation the search is carried out by applying a local search on a candidate solution with the best improvement strategy. The best neighbor of the solution which is either not tabued, or tabued but fulfils the aspiration criterion, is selected. This selected neighbor is not necessarily better than the current solution. This strategy enhances the probability for the search to escape from a local optimum. With

respect to the short term memory, a long term memory also known as the frequency memory is adopted to record the frequencies of being visited for different regions in the search space. The frequency memory helps to diversify the search by directing the search to those least visited regions. This diversification can be applied after the search stagnates for a certain number of generations.

```
Procedure:: Tabu Search (working population)
BEGIN
   Get the optimal solution s_best from input
   Initialize the size of tabu list
   Initialize the best solution found s_bestFound with the best solution in the working population
   While (stopping conditions are not satisfied)
   Begin
      For each solution s_i in working population
      Begin
         Initialize the best neighbor s'_bestNeigh with the largest integer available
         Initialize the best improvement value Δ_best := -Maximum
         For each neighbor s' of solution s_i
         Begin
            If neighbor s' is not tabued And (s_i - s') > Δ_best, then
               Update the best neighbor s'_bestNeigh := s'
               Update best improvement value Δ_best := (s_i - s')
            Else If neighbor s' is tabued but fulfill the aspiration criterion (s_i - s') > 0, then
               If (s' - s_i) > Δ_best, then
                  Update the best neighbor s'_bestNeigh := s'
                  Update best improvement Δ_best := (s_i - s')
               End If
            End If
         End For
         Update the solution s_i := s'_bestNeigh
         Decrease the tabu lengthes for all tabued moves on tabu list by 1
         Update the move (s_i, s') into tabu list
      End For
      Update the best solution found s_bestInPop with the best solution in current working population
      If (s_bestInPop < s_bestFound), then
            s_bestFound := s_bestInPop
      End If
   End While
End
```

Figure 2.6 Pseudo-code of a tabu search

Robust tabu search was proposed by Taillard (1991). It is a popular method for optimization problems because its simplicity and high efficiency for different

applications. In this metaheuristic method, the tabu tenure varies in a uniform random manner over a predefined interval. A long term memory is used in robust tabu search to diversify the search by forcing certain moves on the least visited region periodically. Based on the structure of robust tabu search, Battiti and Tecchiolli (1994) introduced the reactive tabu search. Within reactive tabu search the tabu tenure is adapted to the search situation. When the search returns to the solution previously visited, the tabu tenure increases; while the tabu tenure decreases if it is not changed for a certain number of iterations. The applications of tabu search are quite widespread. The book by Glover and Laguna (1997) presents the fundamental concepts of tabu search and a wide range of applications such as vehicle routing problem, TSP, QAP and scheduling problems. The pseudo-code for the TS is given in Figure 2.6. The pseudo-code shown assumes that TS is applied on a minimization problem.

2.4.9 Swarm Intelligence

Different from the evolutionary algorithms, the swarm intelligence methods are inspired by the social collective behavior of species such as ants, birds, fishes, bees, etc. Usually there is no central control structure inside a typical swarm intelligence method. Some relatively independent and simple agents carry out the search, at the same time they interact with each other through some medium or operating space, leading to a group intelligent behavior. The two most well-known swarm intelligence methods are the ant colony optimization and the particle swarm optimization. Details of the two techniques are presented next.

2.4.9.1 Ant Colony Optimization

Ant colony optimization (ACO) is a swarm intelligence method inspired by the specialized behavior of biological ant colony. It was first proposed by Dorigo (1992). In real world, an ant colony builds up an efficient way of finding food sources. Individual ants have the ability to share knowledge with each other. In the beginning, each ant wanders around randomly, once a food source is located by an ant, that ant returns by the same path and leaves a pheromone trail along the way. The pheromone starts to evaporate right after it is laid. The shortest path to the food source will have the highest density of pheromone since the pheromone left by ants on this path has the shortest time to evaporate. The ants choose a path based on the pheromone level on that path. The denser the pheromone left on a path, the larger the probability that the path will be chosen. More ants will choose the shortest path because of its highest density of pheromone and leave more pheromone on this path. By means of this simple communication mechanism through the pheromone trails, an ant colony is capable of finding the shortest path to a food source efficiently and effectively.

ACO simulates the collective behavior of ants in the biological ant colony. This optimization method consists of two major processes at every generation: the stochastic solution construction and the pheromone update. The pheromone model (trails) is the most important component of the ACO. It is composed of parameters associated with all the elements of a solution. The greedy randomized construction phase is an iterative process of adding solution elements to a partial solution by an artificial ant. Pheromone

trails with the experience of generating good solutions provide guidance for the artificial ants in constructing solutions. Sometimes some problem-dependent heuristic information can also be adopted to help construct a better solution. The pheromone update process consists of two steps: the evaporation and the update. In the evaporation phase, the pheromone parameters are reduced based on a reduction rate. A high reduction rate leads to a more randomized construction phase and more diversification of the search. The update phase can be carried out based on different strategies. The on-line update refers to the update which is carried on even if there are still uncompleted solutions being constructed by artificial ants (Maniezzo 1999). Another popular strategy is to carry on the update after every artificial ant completes the construction of a solution, also named as off-line update. Different selection methods are available for the off-line update to select solutions for applying the update. One method is to select the best k solutions from the total n solutions ($k < n$) (Dorigo et al. 1996). The pheromone parameters are updated with values proportional to the corresponding solution values. The selection of only the best solutions encourages the intensification of the search. Sometimes the worst solution can be selected intentionally to decrease the intensification level (Cordon et al. 2000). The search repeats the construction and the pheromone update until some criterion is satisfied. ACO is applied to solve different optimization problems such as QAP (Stützle and Dorigo 1999), vehicle routing problem (Gambardella et al. 1999b), scheduling problems (Merkle et al. 2000), etc. In many applications ACO hybridized with the local search methods yields excellent results, such as in (Stützle and Hoos 1997). Local search optimizes the solutions constructed by the artificial ants and the refined solutions are

used to update the pheromone trails.

2.4.9.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is another swarm intelligence method which is designed to model the social behavior of natural organism groups, for example, swarm of bees, flock of birds. PSO was first introduced by Kennedy and Eberhart (1995). Inside a typical organism group, each individual has its own target and the corresponding movement to achieve the target without a central coordinator.

Inside a typical PSO method, a set of particles are distributed in a hyperspace modeling the search space for a problem. Every particle in the swarm is a candidate solution for the problem and has its own position and velocity. The positions or coordinates of all particles are adjusted iteratively towards the optimum according to their velocity vectors. Neighbors for a particular particle can be defined as the whole set of particles or just a subset of particles which can be connected to that particle structurally. The PSO with the first neighborhood structure is known as the global best method. The second one which utilizes the subsets is known as the local best method. The velocity vector of one particle is adjusted with reference to its old velocity vector, the inertia of the particle, the global best position visited by the whole population of particles (or the local best position visited by this subset of particles) and the best solution visited by that particle. Through this adjustment, an individual particle shares information with its neighbors. If a neighbor of a particle is in a better position, this particle tries to be closer to that neighbor. The particles in one subset tend to become similar to each other and the particles belong to

different subsets tend to be different. Usually different subsets produce different local optima.

2.4.10 Neural Network

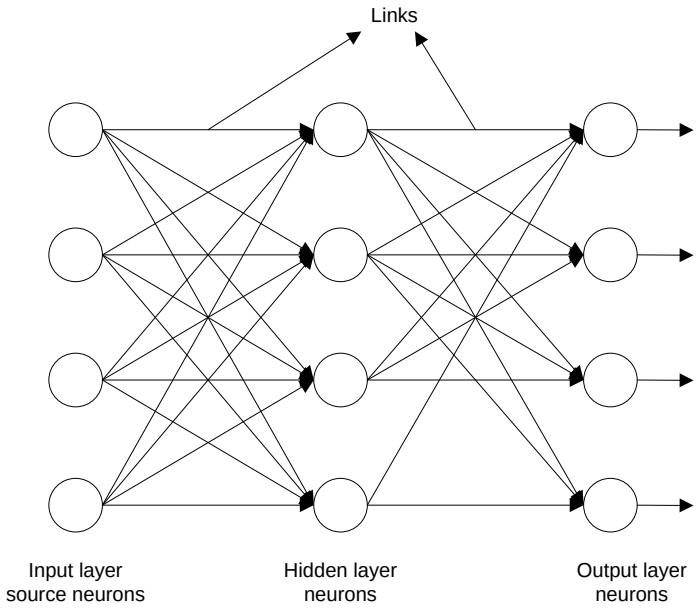


Figure 2.7 A feed-forward neural network

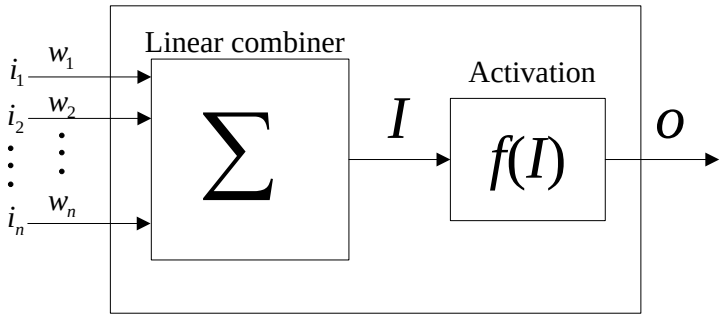


Figure 2.8 The structure of a neuron with an adder and an activation function

The neural network in computational intelligence is also referred to as “artificial neural network” (ANN) in order to distinguish it from the “biological neural network” in biology. ANN is a simplified computational model to simulate the biological neural

networks. In 1943, McCulloch and Pitts (1943) showed the learning capability of ANNs for the first time.

An ANN consists of interconnected artificial neurons, also named as processing elements or nodes. In Figure 2.7 a typical feed-forward neural network is presented. Similar to the biological neural network, the artificial neurons in an ANN perform certain task collectively. Each neuron has one output and one or more inputs. In Figure 2.8 a simplified model of a neuron with a linear combiner as the operator and an activation function is depicted. Each connection link between two neurons has a weight value used to multiply the signal transmitted on that link. During the learning process the structure or the parameters of an ANN changes based on some given information to adapt the ANN to a certain application. The inputs to a neuron are preprocessed by an operator. One popular operator is the adder or linear combiner which performs linear combination of the inputs. Other operators are also available, such as the multiplier which outputs the product of the inputs. Output from the preprocessing operator undergoes an activation function. Examples of activation functions include sigmoidal, hyperbolic tangent, linear, binary, and many others. The adder operator together with a nonlinear activation function is also known as the *nonlinear weighted sum*.

ANNs can be divided into two groups in terms of their structures. One is the feed-forward networks within which the signals are transmitted from input layer to output layer in a unidirectional way. The other is the recurrent networks in which at least one output of a neuron is used as a feedback input to any neuron in its own layer or

preceding layers. Furthermore, feed-forward networks can be classified as partially connected or fully connected networks. In a fully connected network, all neurons from one layer have connections with all neurons in the next layer, which is not necessary in a partially connected network. A partially connected network is usually adopted when the prior knowledge is known for the relation between its input and output. A partially connected network is relatively easy to be trained since it has fewer weight values compared with the fully connected network with the same number of layers and neurons. Examples of feed-forward neural networks include the learning vector quantization network (Kohonen 1988), the multi-layer perceptron (Cybenko 1989). In recurrent networks signals are transmitted in a bidirectional way. The outputs of a recurrent network at a point in time reflect both the current inputs and the previous inputs and outputs. This is the dynamic aspect of the recurrent networks. Hopfield network (Hopfield 1982) is a well-known recurrent neural network which adopts a dynamically stable configuration to store and retrieve patterns for complete or incomplete input patterns.

The most significant feature of an ANN is the capability of learning. For an ANN with a given task and an associated environment (for example, some designated input and output pairs), learning (or training) is a continuous process of adjusting the parameters of the ANN with some method in the environment to achieve a desired configuration with which the ANN can solve the task optimally. There are three major learning methods, the supervised learning, the unsupervised learning and the reinforcement learning. During a

supervised learning, a given input is applied to the network and the network output is compared to a desired output. The difference between the desired and the actual network output is used as an error signal to direct the adjustment of the weights. This process can be executed iteratively with multiple given input and desired output pairs until the network outputs match those desired outputs. This learning requires a "supervisor" to provide the desired input and output pairs. Examples of the supervised learning include the back-propagation rule (Werbos 1974, Rumelhart et al. 1986), delta algorithm (Widow and Hoff 1960). In contrast to the supervised learning, the unsupervised learning does not require a supervisor, i.e. no desired outputs are needed. During the training, the given input patterns are applied to the network. The weights of the network are adjusted to cluster the input patterns into a certain number of groups with common features. With the reinforcement learning, a critic is employed to evaluate the network outputs for some given inputs. The weights are adjusted to achieve better evaluation results.

In http://tralvex.com/pub/nap/ a wide range of applications of ANNs are elaborated, such as pattern detection, signal filtering, virtual reality, data segmentation, data compression, data mining, text mining, adaptive control, optimization and scheduling, etc. In (Haykin 1999), all the important aspects of ANNs are thoroughly investigated with examples.

2.5 Memes and Memetic Computing

In the book *the selfish gene* by Dawkins, the term meme was coined and defined as "a unit of cultural transmission, or a unit of imitation". With his description, the relation of meme with cultural evolution is analogue to the relation of gene with natural evolution (Dawkins 1976). With the growing popularity of the idea of meme since it was introduced, the new Oxford English Dictionary defined it as "an element of a culture or system of behaviour passed from one individual to another by imitation or other non-genetic means."

In the book *the meme machine* by Susan Blackmore, the ideas and applications of meme are further discussed systematically. In her book, a meme is classified explicitly as an evolutionary replicator. The three essential features of an evolutionary replicator are variation (or mutation), selection and retention (or heredity) (Blackmore 1999). Before the introduction of meme, gene is the only evolutionary replicator on earth (Dawkins 1976). When a meme is passed from one individual to another, for example, a song is taught by one person to another, there must be some difference between the two persons when they sing the same song. This fits the concept of variation. As for the selection rule, for example, the popularity of two clothes fashions are different, the one which is more popular is selected by people and survived, the other which is not that popular gradually loses its significance. When a meme is passed on from one person to another, most or some parts of the original meme are retained via learning or imitation. This fits the

concept of retention. In this way, a meme undergoes replication just as what a gene is subjected to in evolutionary processes according to the definition of replicator by Dawkins. Compared to genes which are in the form of DNA molecules, memes are "instructions embedded in human brains, or in artifacts such as books, pictures, bridges or steam trains" (Blackmore 1999). Memes are transmittable between individuals. Examples of memes include languages, ideas, concepts, skills, tunes, clothes fashions, etc. The cultural evolutions of different aspects of human society are realized through the transmission and competition of different memes, for example, the evolution of human language (Blackmore 1999), improvement of technologies.

The memetic or cultural evolution is arguably treated as a soft inheritance (Hull 1982, Dennet 1991, Gould 1991). Soft inheritance or Lamarckian inheritance is an evolutionary theory claims that an organism is capable of passing the characteristics it acquired during its life time to its offspring. This theory is named after the French biologist Jean-Baptiste Lamarck. In modern evolutionary synthesis based on the natural selection theory by Charles Darwin (Darwin 1859), the soft inheritance theory is abandoned. Today it is widely accepted that the phenotype of an organism is decided by its genotype which is the total make-up of its genes, not the other way around. For memetic evolution, the memes as cultural units are almost always modified to a certain level during their transmission from one person to another, for example, a story is told by A to B, while when B tells the same story to C, the story will not be the exact version as told by A. Such modifications for memes are cumulative. Despite Dawkins and Blackmore, other

researchers also give different descriptions of meme, such as “constellations of activated and non-activated synapses within neural memory networks”, “arrays of modified synapses” (Delius 1989), “ideas, the kind of complex idea that forms itself into a distinct memorable unit” (Dennet 1991), “memory item, or portion of an organism’s neurally-stored information” (Lynch 1991), “the least unit of socio-cultural information relative to a selection process that has favourable or unfavourable selection bias that exceeds its endogenous tendency to change” (Wilkins 1998), “a symbolic representation of any state of affairs” (Aunger 2000) and “a different token of the same type of entity as a gene” (Distin 2005), etc.

The term “memetic algorithm” (MA) was first introduced in 1989 by Moscato (1989) to describe the traditional evolutionary algorithm hybridized with a local refining method. Since then many meme-inspired search methods were raised by researchers in recent two decades. In 2009, the journal Memetic Computing was launched to specifically give concentration on the importance of hybrid search methods which are often named as memetic algorithms (Lim et al. 2009). The term “memetic computing” was defined in (Ong et al. 2010) as “a paradigm that uses the notion of meme(s) as units of information encoded in computational representations for the purpose of problem solving.”

In (Meuth et al. 2009), the development of MAs is systematically divided into four generations according to the level of applying the idea of memetic evolution to the optimization methods. The first generation MAs are simply the hybridization of evolutionary algorithm and a local search method, commonly referred to as MA, as

coined by Moscato in 1989. The MAs in this generation are criticized to be missing the core features of the memetic evolution: selection, variation and heredity. With this reason the MAs in the first generation are hardly to be considered as successful applications of meme theory in computer science. The MAs in the second generation refer to the methods which exhibit some of the memetic evolution features such as selection and heredity. Examples of the second generation MAs include hyperheuristics (Kendall et al. 2002), multimeme (Krasnogor et al. 2002), meta-Lamarckian (Ong and Kean 2004, Ong et al. 2006). Within these methods the memetic evolution features are realized. In the multimeme method, multiple local searchers are available, memetic material defining which local searcher to use is decoded in every individual. The individuals are performing local searches accordingly and the memetic materials are transmitted from parents to their offspring individuals. With the transmission of memetic materials, the heredity aspect is achieved. In the hyperheuristic method, the local search methods modeled as memes are selected and applied on the solutions based on some strategies relating with the characteristics of the memes and the current search region. In the meta-Lamarckian method, different local search methods are modeled as memes and compete with each other based on their reward values generated by a reward system with their past performance. The meme with higher reward value is selected to perform the corresponding local search. Detailed reviews on the second generation MAs are given in (Ong et al. 2006). The MAs of the third generation fully exhibit the three memetic evolution features, such as the self-generation MAs (Krasnogor and Gustafson 2004) and the co-evolving MAs (Smith 2007). With the self-generation MA, the local search

strategies modeled as memes are evolving in memetic level with the changing of the search situation. While in the co-evolving MA, a local search operator (LSO) is governed by rules such as the depth of applying the LSO, the fitness of the LSO (related to the past performance of the LSO). The rules are represented in bit strings and undergo operators such as recombination, variation. This rule-based model evolution actually simulates the memetic evolution with the LSO modeled as a meme. More details on the two types of MAs can be found in (Nguyen et al. 2008).

With the summary of the three generations of MAs, the definition of meme in computational intelligence is given in (Meuth et al. 2009):

> "Memes can be seen as mechanisms that capture the essence of knowledge in the form of procedures that affect the transition of solutions during a search."

In almost all the MAs, the objective of manipulating different memes is to strike a healthy balance between intensification (or exploitation which is usually achieved by global search methods) and diversification (or exploration which is usually achieved by local search methods). Furthermore in (Meuth et al. 2009) a brain inspired meta-learning memetic computational system is classified as the fourth generation of MA. This system consists of an optimizer, a memory, a selection mechanism and a generalization mechanism that conceptualizes memes in a more generic contextual scope instead of within the scope of a problem instance. The memory component is to store the generalized patterns of previously solved problems. The features of problem instances are used to direct the construction of meta-memes through a meta-learning process. The

selection mechanism is responsible for storing the constructed meta-memes and performing association between the features of the current problem under test and the previous generalized patterns to achieve high-quality results. In (Ong et al. 2010), the details about the development history and the future possible research fields of MAs are thoroughly investigated.

2.6 Summary

In this chapter, an overview is given on the combinatorial optimization problems and some popular metaheuristic methods. In particular, the QAP which is chosen as the test problem set for the work presented in this thesis has been discussed in details.

The review of metaheuristic methods shows that the main objective of different metaheuristic methods is to keep a healthy balance between exploration and exploitation. Different methods adopt different strategies to achieve this target. For this purpose, hybrid methods are generally more robust since it allows for two or more metaheuristics/heuristics as compared to methods that rely on a single strategy. Therefore, the search by hybrid methods can go further and deeper.

Furthermore, the development of the meme and memetic theory is discussed in details. The meme and memetic theory emphasize on the utilization of the linkages between different problems and different search methods. The theory provides the theoretical

basis for the work presented in this thesis.

Chapter 3 Collaborative Memes Framework (CMF)

3.1 Introduction

The work presented in this chapter represents an open configuration for managing or coordinating multiple memes or search methods within a collaborative memes framework. Although in the work reported here, three specific memes are specified, the proposed scheme is flexible to include any other memes. This is unlike the island model type of search methods where the focus is on partitioning the population into subgroups in order to stimulate population diversity. It is also different from the general scheme of hybridization as in the case of typical hybrid evolutionary methods. In hybridization, two or more search methods which ideally work in a complimentary manner are combined together. Furthermore, the combination of the methods is structured such that the flow of the search process occurs in a sequential manner, for example, the genetic algorithm and the *k*-gene exchange local search in the hybrid genetic algorithm (Lim et al. 2002). Within the proposed collaborative memes framework (CMF), memes are allowed to combine in a very flexible manner, without the needs for matching of parameters.

Population diversity is maintained through independent population samples coupled with a communication protocol that is designed to allow the memes to share their results so as to promote convergence towards the optimum solution.

Here the goal is not exactly to put forth an approach that will consistently outperform any of the techniques reported in the literature. Rather, the work here is an attempt to draw meaningful conclusions on the potential merits of constructing a collaborative memes approach for multiple optimization methods. In particular, stochastic or approximate search methods that in recent years have gained immense popularity are suitable to be studied with. The effect of this collaborative strategy on search performance will be shown by means of experimental results. In order to accomplish this, several issues worthy of further research are highlighted.

In Section 3.2, the details of the collaborative memes framework used to solve the QAP are discussed. In Section 3.3, the communication protocol which regulates the sharing of search results between memes is discussed. In Section 3.4, the method of experimentations and present simulation results on groups of large-scale QAP benchmarks is described; additionally some discussions on the experimental results are presented. In Section 3.5, a summary is presented for this chapter.

3.2 Collaborative Memes Architecture

Over the years, many different optimization methods have been developed for solving complex optimization problems (Skorin-Kapov 1990, Laursen 1993, Maniezzo et al. 1994, Taillard 1995, Tate and Smith 1995, Ishii and Sato 1998). In recent years, methods incorporating multiple memes typically in the form of a global search such as population based approaches like genetic algorithms coupled with a localized search methodology have become very common (Merz and Freisleben 1999, Merz and Freisleben 2000, Lim et al. 2000, Lim et al. 2002, Ong and Keane 2004). From the experience of dealing with evolutionary algorithm-based meta-meme approaches in solving combinatorial optimization problems, one can conclusively state that no one single method can consistently outperform all other search methods, a premise that is consistent with the "*no free lunch theorem*" in optimization. Hence, in the context of QAP, it is clear that one would be hard pressed to come up with an approach that can consistently outshines all other approaches on every single problem instance when tested on a large collection of benchmarks that are commonly used in the QAP community.

An instantiation of CMF is demonstrated by incorporating three different memes, simulated annealing (SA), tabu search (TS) and genetic algorithm (GA). Each meme maintains its own population of solutions. The details of the three memes have been discussed in Sections 2.4.1, 2.4.7 and 2.4.8. After each generation, the populations of different memes exchange individual(s). In the proposed scheme, inter-population

transfer or migration of individuals is an important communication channel that further promotes exploitation and population diversity. Although a great deal of flexibility can be exercised in this aspect, care must be taken to prevent over-dominance of any one meme in the collaborative framework. As a general rule, the communication protocol for regulating the exchange of individuals between each pair of memes in the CMF should strike a healthy balance between exploitation and population diversity.

Within the CMF, different memes can serve to complement each other, henceforth giving rise to a more consistent search performance when tested on several suites of QAP benchmarks. In principle, CMF has the capacity to perform to a level that is as good as or better than any of the meme independently within the scheme on every problem. The proposed CMF is illustrated in the flow diagram shown in Figure 3.1. Starting with the initialization of the population pools, GA, SA and TS carry out search independently from Step 4 to Step 6. The search results for each of the method are preserved within every population pool. In Step 7, three population pools exchange individual solutions based on the communication protocol described in Section 3.3. After the exchange of solutions, the three methods carry out the search for the next generation with the updated population pools. The sharing of the search results guarantees the hybridization of the three methods and results in better solution quality compared to that of the individual methods running independently. Through the communication protocol, a healthier balance is achieved between the diversification (mainly contributed by GA) and the intensification (mainly contributed by SA and TS) compared to the individual methods.

Therefore, the search efficiency and the solution quality of the three methods with CMF are improved.

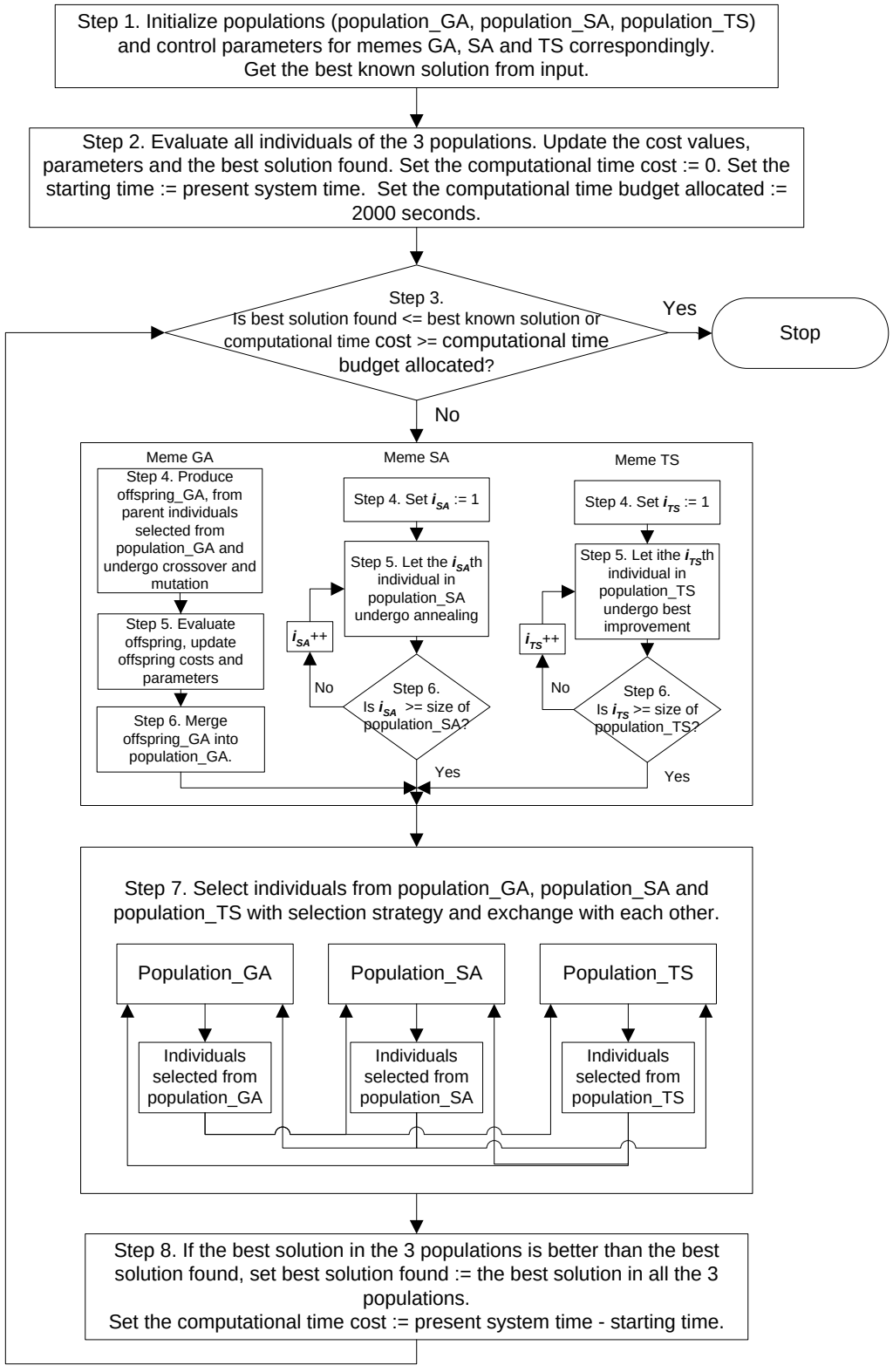


Figure 3.1 Flowchart of the CMF (For more details about the three memes included, refer to Sections 2.4.1, 2.4.7 and 2.4.8)

The concept of managing multiple memes collectively is therefore logically consistent with observations of earlier attempts to configure hybrid methods to solve the QAP (Merz and Freisleben 1999, Merz and Freisleben 2000, Lim et al. 2000, Lim et al. 2002, Ong and Keane 2004, Tang et al. 2006). By configuring a scheme that links together multiple memes, one can synergize the search capacity of those memes giving rise to an overall performance improvement when compared to a single well-tuned meme. Such a scheme that incorporates and manages various memes is referred to as a collaborative memes framework (CMF).

3.3 Communication Protocol

In CMF, three independent populations are configured with a communication protocol for regulating the sharing of search results through a pre-established immigration policy. A robust communication protocol is crucial to create a conducive population dynamics that promotes exploitation and exploration. In the beginning, a method of "replace the worst by the best" was adopted, in other words, only the best individuals are chosen from the source populations. The chosen individuals are used to replace the worst individuals in the destination populations unconditionally. With such a protocol, the framework did exhibit superiority over each independent meme. However, there were instances whereby the framework failed to outperform the independent memes. One possible reason why this protocol did not work as expected is that this absolute strategy has the

tendency of destroying the diversity of all populations resulting in premature convergence of the search. Another protocol used is the "randomly choose and replace". The individuals are randomly chosen from a source population. These individuals are merged into a destination population by replacing the same number of randomly chosen individuals in that destination population. This method has proven to be difficult to control based on simulation results on the QAP benchmarks. After further investigations, the following communication protocol was finalized.

For each generation of CMF, the search process is carried out independently in every population. Each population pool accepts individual(s) selected from the other two population pools and together with its own population form a new population. The main issues to consider in the migration of individuals among the population pools are described next.

Stagnancy - An individual is labeled as being in a state of stagnancy if it is not improved for a fixed number of generations. Such individuals which stagnate will be targeted for replacement by individuals immigrating from the other two populations.

Prohibition of immigration - After an individual is selected from one source population and successfully merged into another pre-destined population, this individual will be prohibited from migrating between these two populations for a certain number of generations. This is to reduce the possibility of poor diversity in all the working populations.

Selection of individuals – A certain number of best individuals will be selected from two source populations to form the immigrants for the third population pool. The individual(s) selected should not be in *prohibition of immigration* state specifically for immigration between the source and destination populations.

Replacement into destination population - The individuals in a destination population pool will be examined in descending order in terms of their solution costs. If an individual is found to be in *stagnancy* state, it will be replaced by an individual from the immigrant population. This replacement process will be carried out until all the individuals in the immigrant population pool are replaced into the destination population or all the individuals in the destination population pool are examined and replaced if the stagnation condition is satisfied.

Drastic perturbation (Stützle 2006) - An individual undergoes drastic perturbation if it is in *stagnancy* state for a number of generations which reaches or exceeds the preset maximum number of generations allowed for an individual to remain in *stagnancy* state and without applying perturbation. Drastic mutation refers to *k* random moves applied to an individual. The parameter *k* is configurable in the beginning. It is chosen to be related to the size of the problem being solved. The perturbation operator is applied to all individuals in the populations associated with memes GA, SA and TS running independently as well as the CMF.

3.4 Experimental Results and Analysis

3.4.1 Computational Platform and Parametric Settings

The instantiation of the CMF configuration consists of three populations, each of which is associated with one of the three memes: simulated annealing, tabu search and genetic algorithm respectively. The scheme was tested and its performance was compared against that of each individual meme executing independently. These search methods were coded in C++. The computing platform in the simulations is a Pentium 3.4GHz CPU with 2GB memory in Windows XP operating system. The test problems consist of 37 large-scale QAP benchmarks. For each problem, 10 separate runs were carried out for every method. For each run, a computational time budget of 2000 seconds is allocated.

The parametric settings of the 3 independent memes are as follow:

--*Tabu Search*: population size = 60; tabu length = 6; number of generations to ascertain whether an individual is in stagnancy state = 2,

--*Simulated Annealing*: population size = 60; initial temperature = 30.0; temperature annealing factor α = 0.9998; number of generations to ascertain whether an individual is in stagnancy state = 2;

--*Genetic Algorithm*: population size = 60; crossover rate = 1.0; mutation rate = 0.05; mutation range = 0.01; number of generations to ascertain whether an individual is in

stagnancy state = 3.

The other parameter settings that are common among all three memes are as follows:

Perturbation range k = 0.3*(problem size); maximum number of generations allowed for an individual to remain in stagnancy state and without applying perturbation = 10.

The configuration of the CMF is as follow:

Population size of GA population = 20; population size of SA population = 20; population size of TS population = 20; tabu length for immigration from GA population = 3; tabu length for immigration from SA population = 6; tabu length for immigration from TS population = 6; and the maximum number of individuals to be selected from each population for immigration purpose = 5.

3.4.2 Experimental Results

Tables 3.1 to 3.4 summarize the test results on 37 benchmarks by GA, SA, TS and CMF. The results are presented by separating the benchmarks into 4 categories: *lipaxx* in Table 3.1, *skoxx* in Table 3.2, *taixx* in Table 3.3 and finally the miscellaneous benchmarks in Table 3.4. For each benchmark, the best performance among all the four search methods is highlighted in bold. The *Benchmark* column lists the benchmark names and the optimum solutions, digits tagged to each name indicate its problem size. *Method* refers to the corresponding search method. The heading *CPU time* refers to the average computational time cost for all 10 runs.

Table 3.1 Results of testing on QAP benchmarks *lipaxxx*

Benchmark	Method	CPU time(sec)	Average performance	Average gap(%)	Best performance	Best gap(%)	Success rate(%)
Lipa50a 62093	GA	1514.3	62804.2	1.15	62767	1.09	0
	SA	710.0	62303.8	0.34	**62093**	**0.00**	50
	TS	1015.6	**62093.0**	**0.00**	**62093**	**0.00**	**100**
	CMF	123.5	**62093.0**	**0.00**	**62093**	**0.00**	**100**
Lipa50b 1210244	GA	1477.3	1405520.3	16.14	**1210244**	**0.00**	10
	SA	9.8	**1210244.0**	**0.00**	**1210244**	**0.00**	**100**
	TS	16.2	**1210244.0**	**0.00**	**1210244**	**0.00**	**100**
	CMF	8.4	**1210244.0**	**0.00**	**1210244**	**0.00**	**100**
Lipa60a 107218	GA	1694.1	108345.6	1.05	108234	0.95	0
	SA	1338.0	108043.1	0.77	107967	0.70	0
	TS	1287.5	108036.7	0.76	107970	0.70	0
	CMF	705.0	**107359.3**	**0.13**	**107218**	**0.00**	**80**
Lipa60b 2520135	GA	1632.6	3021107.4	19.88	3003732	19.19	0
	SA	57.6	**2520135.0**	**0.00**	**2520135**	**0.00**	**100**
	TS	72.6	**2520135.0**	**0.00**	**2520135**	**0.00**	**100**
	CMF	32.4	**2520135.0**	**0.00**	**2520135**	**0.00**	**100**
Lipa70a 169755	GA	1811.0	171502.3	1.03	171385	0.96	0
	SA	1501.8	170967.9	0.71	170891	0.67	0
	TS	1310.6	170934.3	0.69	170885	0.67	0
	CMF	418.7	**170093.7**	**0.20**	**169755**	**0.00**	**70**
Lipa70b 4603200	GA	1696.0	5588901.5	21.41	5572069	21.05	0
	SA	110.8	**4603200.0**	**0.00**	**4603200**	**0.00**	**100**
	TS	84.6	**4603200.0**	**0.00**	**4603200**	**0.00**	**100**
	CMF	56.6	**4603200.0**	**0.00**	**4603200**	**0.00**	**100**
Lipa80a 253195	GA	1693.5	255715.1	1.00	255552	0.93	0
	SA	1685.2	254850.1	0.65	254778	0.63	0
	TS	952.1	254831.6	0.65	254793	0.63	0
	CMF	879.5	**254259.9**	**0.42**	**253195**	**0.00**	**30**
Lipa80b 7763962	GA	1502.3	9549924.9	23.00	9504159	22.41	0
	SA	250.3	**7763962.0**	**0.00**	**7763962**	**0.00**	**100**
	TS	331.8	**7763962.0**	**0.00**	**7763962**	**0.00**	**100**
	CMF	172.2	**7763962.0**	**0.00**	**7763962**	**0.00**	**100**
Lipa90a 360630	GA	1732.0	364158.0	0.98	364006	0.94	0
	SA	1741.4	362877.3	0.62	362821	0.61	0
	TS	1092.5	362811.6	0.60	362767	0.59	0
	CMF	1102.1	**362239.6**	**0.45**	**360630**	**0.00**	**20**
Lipa90b 12490441	GA	1627.7	15448914.0	23.69	15410660	23.38	0
	SA	368.6	**12490441.0**	**0.00**	**12490441**	**0.00**	**100**
	TS	466.6	**12490441.0**	**0.00**	**12490441**	**0.00**	**100**
	CMF	254.3	**12490441.0**	**0.00**	**12490441**	**0.00**	**100**

Average performance refers to the average solution value. *Average gap* refers to the percentage difference between the average solution and the optimum solution. *Best performance* refers to the best solution found in all 10 runs. *Best gap* refers to the percentage difference between the best solution found and the optimum solution. *Success rate* refers to the percentage of achieving the optimum solution in 10 runs.

Table 3.2 Results of testing on QAP benchmarks *skoxxx*

Benchmark	Method	CPU time(sec)	Average performance	Average gap(%)	Best performance	Best gap(%)	Success rate(%)
Sko56 34458	GA	1429.3	35146.2	1.99	34810	1.02	0
	SA	998.8	34507.4	0.14	34482	0.07	0
	TS	924.8	34480.2	0.06	34462	0.01	0
	CMF	589.7	**34464.2**	**0.02**	**34458**	**0.00**	**60**
Sko64 48498	GA	1836.2	49567.2	2.20	49306	1.67	0
	SA	1310.0	48604.6	0.22	48578	0.16	0
	TS	1078.8	48546.6	0.10	48510	0.02	0
	CMF	414.0	**48501.2**	**0.01**	**48498**	**0.00**	**80**
Sko72 66256	GA	1838.1	68095.4	2.78	67616	2.05	0
	SA	1235.0	66431.2	0.26	66340	0.13	0
	TS	1145.8	66354.8	0.15	66320	0.10	0
	CMF	773.2	**66339.4**	**0.13**	**66256**	**0.00**	**40**
Sko81 90998	GA	1826.9	94006.2	3.31	93516	2.77	0
	SA	1617.4	91233.6	0.26	91186	0.21	0
	TS	1214.5	91146.0	0.16	91114	0.13	0
	CMF	1264.0	**91038.2**	**0.04**	**90998**	**0.00**	**10**
Sko90 115534	GA	1877.1	120471.8	4.27	119460	3.40	0
	SA	1746.4	115900.8	0.32	115800	0.23	0
	TS	1682.6	115850.2	0.27	115770	0.20	0
	CMF	1240.0	**115666.8**	**0.11**	**115534**	**0.00**	**10**
Sko100a 152002	GA	1848.0	159163.6	4.71	157730	3.77	0
	SA	1601.1	152524.8	0.34	152442	0.29	0
	TS	1514.6	152483.0	0.32	152390	0.26	0
	CMF	1359.0	**152137.8**	**0.09**	**152002**	**0.00**	**10**
Sko100b 153890	GA	1859.4	160979.6	4.61	159906	3.91	0
	SA	1534.7	154394.8	0.33	154214	0.21	0
	TS	1606.9	154331.2	0.29	154248	0.23	0
	CMF	1224.8	**153968.2**	**0.05**	**153890**	**0.00**	**10**
Sko100c 147862	GA	1886.2	154802.4	4.69	153894	4.08	0
	SA	1802.4	148261.4	0.27	148122	0.18	0
	TS	1795.7	148311.4	0.30	148234	0.25	0
	CMF	1555.2	**147905.4**	**0.03**	**147862**	**0.00**	**10**
Sko100d 149576	GA	1919.8	156283.8	4.48	155484	3.95	0
	SA	1614.9	150137.8	0.38	150050	0.32	0
	TS	1486.5	150082.0	0.34	149966	0.26	0
	CMF	1437.1	**149699.8**	**0.08**	**149596**	**0.01**	**0**
Sko100e 149150	GA	1818.4	156823.8	5.15	155596	4.32	0
	SA	1778.0	149600.8	0.30	149502	0.24	0
	TS	1562.2	149557.2	0.27	149352	0.14	0
	CMF	1178.5	**149223.0**	**0.05**	**149150**	**0.00**	**20**
Sko100f 149036	GA	1871.1	155495.0	4.33	154322	3.55	0
	SA	1413.4	149675.6	0.43	149568	0.36	0
	TS	1729.6	149616.6	0.39	149468	0.29	0
	CMF	1247.9	**149221.6**	**0.12**	**149036**	**0.00**	**10**

In general, for the benchmarks in *lipaxxx* group, the "*a*" type of benchmarks seems to present much difficulty to the search methods tested. However, as can be seen from the results in Table 3.1, the CMF managed to solve all the problems in this group optimally at least in one of the 10 runs.

Table 3.3 Results of testing on QAP benchmarks *taixxx*

Benchmark	Method	CPU time(sec)	Average performance	Average gap(%)	Best performance	Best gap(%)	Success rate(%)
Tai50a 4941410	GA	1276.8	5123766.0	3.69	5105452	3.32	0
	SA	1005.6	5026296.0	1.72	5019810	1.59	0
	TS	1053.7	5016828.0	1.53	5007264	1.33	0
	CMF	877.5	**4980185.3**	**0.78**	**4968180**	**0.54**	**0**
Tai50b 458821517	GA	1061.9	480786795.8	4.79	462459602	0.79	0
	SA	1164.9	458927511.0	0.02	**458821517**	**0.00**	10
	TS	718.8	458883402.3	0.01	**458821517**	**0.00**	20
	CMF	65.7	**458821517.0**	**0.00**	**458821517**	**0.00**	**100**
Tai60a 7208572	GA	1598.5	7467463.2	3.59	7423220	2.98	0
	SA	1227.9	7355216.4	2.03	7337622	1.79	0
	TS	1215.4	7334058.0	1.74	7323238	1.59	0
	CMF	1310.2	**7292220.0**	**1.16**	**7257524**	**0.68**	**0**
Tai60b 608215054	GA	1411.5	634176330.0	4.27	612422462	0.69	0
	SA	876.4	608431870.1	0.04	608228619	0.00	0
	TS	1464.9	608420316.5	0.03	**608215054**	**0.00**	10
	CMF	214.5	**608215054.0**	**0.00**	**608215054**	**0.00**	**100**
Tai64c 1855928	GA	225.4	1859279.2	0.18	1856396	0.03	0
	SA	0.7	**1855928.0**	**0.00**	**1855928**	**0.00**	**100**
	TS	2.7	**1855928.0**	**0.00**	**1855928**	**0.00**	**100**
	CMF	0.5	**1855928.0**	**0.00**	**1855928**	**0.00**	**100**
Tai80a 13557864	GA	1532.5	14308228.0	5.53	14269494	5.25	0
	SA	1088.4	13801959.0	1.80	13784094	1.67	0
	TS	1128.5	13798083.6	1.77	13773748	1.59	0
	CMF	1396.8	**13685894.8**	**0.94**	**13653704**	**0.71**	**0**
Tai80b 818415043	GA	1843.8	879881633.0	7.51	866675879	5.90	0
	SA	1161.2	819181649.7	0.09	818835258	0.05	0
	TS	1260.2	819788568.4	0.17	818933106	0.06	0
	CMF	894.3	**818453234.3**	**0.00**	**818415043**	**0.00**	**70**
Tai100a 21125314	GA	1443.5	22297354.0	5.55	22219034	5.18	0
	SA	922.3	21500804.8	1.78	21470470	1.63	0
	TS	1065.3	21508918.0	1.82	21463392	1.60	0
	CMF	1622.4	**21346564.2**	**1.05**	**21288930**	**0.77**	**0**
Tai100b 1185996137	GA	1877.6	1282010191.3	8.10	1255698104	5.88	0
	SA	954.1	1187865298.0	0.16	1187259553	0.11	0
	TS	1495.1	1188672147.7	0.23	1187334907	0.11	0
	CMF	1378.5	**1186660643.6**	**0.06**	**1185996137**	**0.00**	**10**
Tai256c 44787190	GA	1815.3	45345148.8	1.25	45220012	0.97	0
	SA	1046.1	44857332.2	0.16	44846960	0.13	0
	TS	1346.7	44882780.4	0.21	44872890	0.19	0
	CMF	**1518.2**	**44846075.0**	**0.13**	**44821610**	**0.08**	**0**

Table 3.4 Results of testing on other QAP benchmarks

Benchmark	Method	CPU time(sec)	Average performance	Average gap(%)	Best performance	Best gap(%)	Success rate(%)
Esc64a 116	GA	45.7	**116.0**	**0.00**	**116**	**0.00**	**100**
	SA	986.9	**116.0**	**0.00**	**116**	**0.00**	**100**
	TS	1.5	**116.0**	**0.00**	**116**	**0.00**	**100**
	CMF	0.9	**116.0**	**0.00**	**116**	**0.00**	**100**
Esc128 64	GA	1233.1	67.8	5.94	**64**	**0.00**	30
	SA	1917.8	149.0	132.81	144	125.00	0
	TS	30.3	**64.0**	**0.00**	**64**	**0.00**	**100**
	CMF	13.7	**64.0**	**0.00**	**64**	**0.00**	**100**
Tho40 240516	GA	876.4	246948.4	2.67	243934	1.42	0
	SA	1235.1	240786.4	0.11	**240516**	**0.00**	20
	TS	826.5	240554.2	0.02	**240516**	**0.00**	**50**
	CMF	485.3	**240545.8**	**0.01**	**240516**	**0.00**	**50**
Tho150 8133864	GA	1809.7	8756231.2	7.65	8696010	6.91	0
	SA	1420.8	8165515.0	0.39	8160918	0.33	0
	TS	1810.6	8197803.2	0.79	8187216	0.66	0
	CMF	1564.0	**8149925.2**	**0.20**	**8144730**	**0.13**	**0**
Wil50 48816	GA	936.4	49414.2	1.23	49042	0.46	0
	SA	971.9	48833.6	0.04	48824	0.02	0
	TS	727.1	48823.6	0.02	48816	0.00	10
	CMF	680.9	**48818.4**	**0.00**	**48816**	**0.00**	**70**
Wil100 273038	GA	1911.3	279929.0	2.52	279180	2.25	0
	SA	1644.5	273651.6	0.22	273568	0.19	0
	TS	1615.6	273561.8	0.19	273454	0.15	0
	CMF	1315.8	**273142.2**	**0.04**	**273038**	**0.00**	**10**

3.4.3 Results and Analysis

From the results presented in Tables 3.1 to 3.4, it is clear that CMF produced better or equal "*average*" and "*best*" solutions for all the 37 benchmarks tested compared to its constituent memes independently with the same configurations and computational time budget. In most scenarios, CMF improves the solutions quality significantly with respect to the constituent individual memes. The success rates of CMF are also the highest for most of the benchmarks tested.

Looking at each of the memes independently, it is clear the GA performed the worst. Among the three independent memes, the meme TS seems to perform the best although

the SA in many instances showed comparable performance. What is evident from these comparisons is that, each of the memes operating independently has less of a chance to reach the optimal solution on every problem. With the collaborative memes approach, the search capacity is greatly enhanced. To make the point clearer, let us consider a specific case, the *wil100* benchmark. From the results shown in Table 3.4, all three memes did not manage to find the optimum solution. However, with the CMF which draws on the collective capability of all the three memes, the optimum solution was uncovered and the average performance was also improved significantly. This draws attention to the fact that the benefit of this collaborative memes approach goes beyond that of simply drawing on the individual strength of each of the memes. The coupling of the various memes in the CMF extends the exploratory range and the interplay between the memes through a communication channel produced a desirable diversification enhancing effect. This is clearly beneficial to the CMF resulting in more robust performance compared to each of the memes operating independently.

The test results on CMF are compared with the results of a typical hybridized genetic algorithm (HGA) reported in (Lim et al. 2002). The results given by CMF are obviously better than the HGA in terms of both average and best found solution values. CMF also gives higher success rates on most of the instances compared to HGA. For example, CMF solved *lip80b* and *lip90b* with a success rate of 100%, but HGA failed in solving these two instances to optimality in all 10 runs. For *skoxx* series benchmarks, CMF managed to find the optimum solutions except for *sko100d*, while HGA failed to find the

optimum solutions for all benchmarks in this group. The results are also compared with the results of the multi-island parallel memetic algorithms reported in (Tang et al. 2007). There are several groups of results by different search methods with different number of islands and different methods of controlling local search in (Tang et al. 2007). The results presented by the 2-island parallel memetic algorithm with diversity-based dynamic adaptive strategy (PMA-DLS) are chosen for comparison because the associated algorithm bears the closest resemblance to CMF in terms of computational capacity. For the 6 *sko100x* benchmarks, the results by CMF are better in 4 out of the 6 in terms of the average solutions quality. For example, the average gap for *sko100a* given by PMA-DLS is 0.11%, which is worse than 0.09% given by CMF. CMF also increases the success rates on *sko100a*, *sko100b*, *sko100d* and *sko100f*, only fails on *sko100e* which PMA-DLS gives a higher success rate of 30% compared to the success rate of 20% by CMF. For *wil100*, CMF gives average solution value of 273142.2 and even finds the optimum solution in one run. For the same instance, PMA-DLS gives a worse average solution value of 273147.2 and fails to find the optimum solution in all its trials. For *tai100a,* CMF gives average and best gaps of 1.05% and 0.77% respectively, but PMA-DLS only gives 1.50% and 1.18% correspondingly. In a summary, for all the 8 benchmarks (*tho150* is not included since a different computational time budget is adopted for this instance in (Tang et al. 2007)), CMF outperforms PMA-DLS for 6 of them.

3.5 Summary

In this chapter, a collaborative memes framework for coordinating and managing three well-known memes is presented with implementation and experimental details. Given the same computational time budget, CMF shows better performance than the three memes running independently based on experiments with a set of selected large-scale QAP benchmarks. The experimental results are consistent with the expectation of the benefits that can be gained from a collaborative approach including multiple memes. Although three most common and well-known memes have been chosen in the proposed CMF instantiation in this chapter, it goes without saying that this scheme is versatile enough to include any other optimization methods as memes as well. The comparison of test results with other two robust methods further proves that the instantiation of the CMF incorporating multiple memes is a competitive method for solving QAP. Hence, if the CMF is constructed with more sophisticated optimization methods, the potential for performance enhancement is greater.

In conclusion, CMF is a more consistent and reliable search methodology compared to the individual memes executing independently. In this chapter, a rather straightforward communication protocol is adopted. Clearly, there are many issues pertaining to how individuals within this scheme are transferred between populations associated with all constituent memes. This aspect deserves further investigation and offers great potential for enhancing the performance of this search methodology. From experience gained in

this study, it is reasonable to conclude that the dynamic runtime adaptation of the communication protocol can go a long way to help circumvent the needs for constant tweaking of the parameters governing the inter-population transfer of individuals.

Besides the robustness of the CMF on combinatorial optimization problems, this approach lacks flexibility. Its performance is inconsistent for different types of combinatorial optimization problems since the strategy of managing memes has to be fixed beforehand. In other words, the optimization recipe is static, meaning the meta-meme is fixed or manually tweaked in order to achieve the desired performance. The proposed CMF manages memes with the strategies based on the knowledge about the characteristics of the memes and the linkages between different memes to strike a healthy balance between exploitation and exploration. The knowledge about the linkages between different problems and even the linkages between memes and problems is not utilized in CMF. Such linkages provide the knowledge to utilize the experience acquired from earlier problem-solving sessions to construct different appropriate meta-memes for managing or coordinating different memes to achieve better performance for various combinatorial optimization problems. To achieve a more robust and problem-adaptive search methodology, the neural meta-memes framework is proposed in the next chapter.

Chapter 4 Neural Meta-Memes Framework (NMMF)

4.1 Introduction

Neural Meta-Memes Framework (NMMF) is an optimization framework for managing different search methods modeled as memes, utilizing every meme's best traits to achieve effective combinatorial optimization. Multiple search methods are modeled as memes to form a meme pool. With NMMF, different memes are alternatively applied on the individual solutions within the main population. Every single meme is a developed optimization mechanism and has its own unique specialty which is usually hard to emulate by any other memes. The NMMF has the potential of establishing a robust and flexible problem-solving methodology that is capable of achieving enhanced performance for different combinatorial optimization problems.

Every meme within NMMF is considered a unique knowledge system with its unique search procedure(s). A meme is able to lead the population in a rather predictable manner. Furthermore, a meme tends to be configured or managed in such a way that it is

domain-specific, typically enhanced with some knowledge specific either to the problem, or gained from earlier instances of solving similar problems. In other words, the NMMF manages memes such that a well-balanced mechanism specifically for the problem under test is achieved. In NMMF, each meme is an independent system whereby standard search procedure(s) are applied on the population. It works in a relatively independent manner, only after one meme finishes its search, another follows to continue with the search. In this way, the scheme allows for a greater level of certainty that each meme is made to exploit its full potential. Due to the complementary nature of the memes used in the NMMF, the meta-meme method generated will naturally integrate them in a collaborative way which further magnifies the overall capability of the approach. In general, NMMF takes advantage of similarities between different combinatorial optimization problems. Similarity is estimated by a set of attributes; higher level of similarity leading to correspondingly similar recipes of search mechanisms.

In the next section, the details of the neural meta-memes framework and its main components are presented. In Section 4.3, a brief introduction on the features of QAPs is presented. In Section 4.4, the details of the NMMF specially designed for QAP are discussed. Experimental results and analysis are reported in Section 4.5. Finally in Section 4.6, a summary is given for this chapter.

4.2 Neural Meta-Memes Architecture

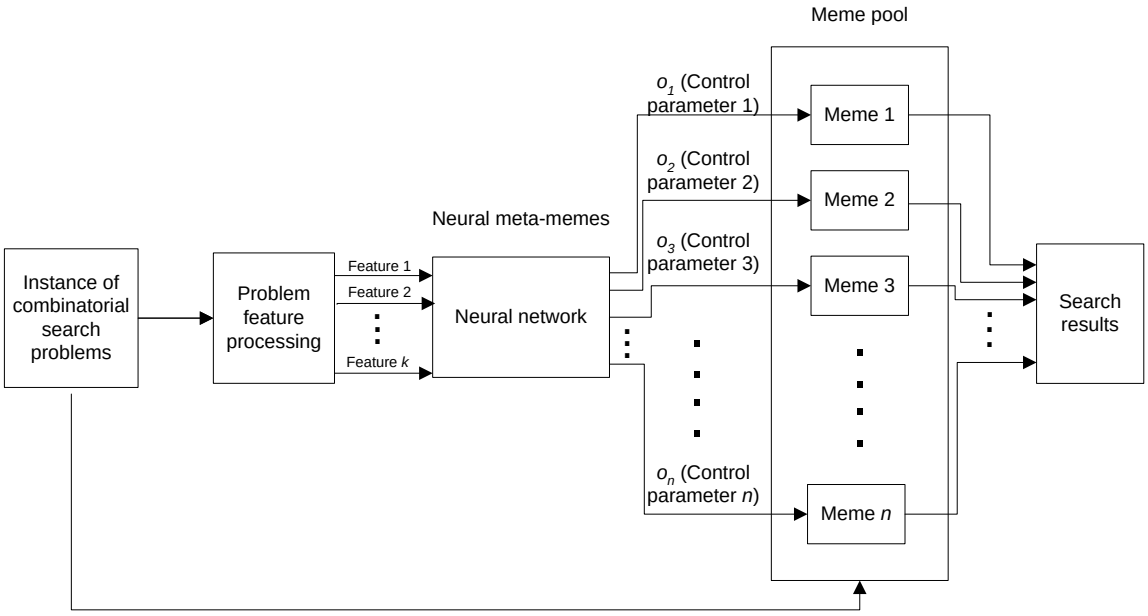


Figure 4.1 Neural meta-memes architecture

A simplified architecture of NMMF is shown in Figure 4.1. Consider a class of combinatorial optimization problems which can be characteristically distinguished by *k* features and a neural meta-memes framework consisting of *n* memes. The procedures of the NMMF in solving combinatorial optimization problems are as follow:

Step 1: The problem data undergo problem feature processing unit to generate the *k* feature numbers.

Step 2: The *k* features serve as inputs to the neural network which is responsible for producing the *n* optimized control parameters for the *n* memes.

Step 3: With the *n* control parameters, the *n* memes carry out their respective search to achieve a synergized search performance.

Here a control parameter for a meme refers to the maximum number of continuous iterations without improvement allowed for that meme. The number of iterations without improvement is indicative of the level of stagnation of a meme.

The simplified flowchart of the NMMF is presented in Figure 4.1 and the flow of the framework is discussed in the above 3 steps. In the beginning, the problem features are extracted from the input problem data through some relatively straightforward calculation procedures. These features contain the information about the problem being presented. The features are inputted to a trained neural network to generate the control parameters for the memes. The neural network contains the knowledge about which memes are suitable and the appropriate level of applying each meme for a certain problem. Therefore, these generated control parameters act as a meta-meme mechanism which can be considered as a specially optimized mechanism for this problem only. With this strategy, NMMF provides a form of guarantee for consistent performance for different types of problems. Finally, all the involved memes carry out the search for the optimum solution of this problem under the control of this generated meta-meme mechanism.

The neural network needs to be trained before it can be used to generate the appropriate control parameters for the memes. The training is analogous to the process of collecting

experience from earlier problem-solving sessions. In the remaining of this section, more details on the generation of the training data and the back-propagation update rule adopted in the training process are presented.

4.2.1 Generation of Training Data

The training data for the neural network consist of features of different problems and the corresponding optimized control parameters. Without loss of generality, the combinatorial optimization problem being considered here is assumed to be a minimization problem. The procedure is as outlined below:

Step 1: Initialize the maximum number of generations allowed to be in stagnation N_{max} to a preset constant and the number of generations in stagnation N_{stag} is initialized to 0.

Step 2: Generate the initial *n* control parameters randomly within some predefined value ranges or simply assign some fixed numbers depending on the characteristics of the corresponding memes.

Step 3: While $N_{stag} \geq N_{max}$, terminate the process, otherwise, perform simulation runs on the input combinatorial optimization problem for different control parameters sets through additive or subtractive adjustment of each parameter.

Step 4: Compare the performance result of each simulation run (i.e., in terms of solution quality and computational time cost) with respect to the best result found. If

the result of a simulation run is better than the best result found, replace the best result found with this result and the present control parameters with the control parameters associated with this simulation.

Step 5: Check whether the present control parameters set has been updated or not in *Step 4*. If it has, set N_{stag} to 0 and go back to *Step 3*, otherwise, increase N_{stag} by 1.

Using the above procedures, the control parameters for the memes are optimized using the relatively small-scale or simple problems as the training examples.

4.2.2 Neural Network Update Rule

The neural network used in NMMF is responsible for generating control parameters for different memes. The inputs of the neural network are the extracted problem features.

The direct outputs from the neural network are the normalized values o'_1, o'_2 … o'_n, hence the actual parameters are obtained as follows:

$$o_x = o'_x \times o_{x\max} \tag{4.1}$$

where o'_x refers to the output of the neural network and o_x refers to the corresponding actual parameter (rounded down to an integer). In the event that the inferred output exceeds its respective pre-defined upper or lower bound, i.e., o_{xmax} or o_{xmin} respectively, the corresponding limit is assigned.

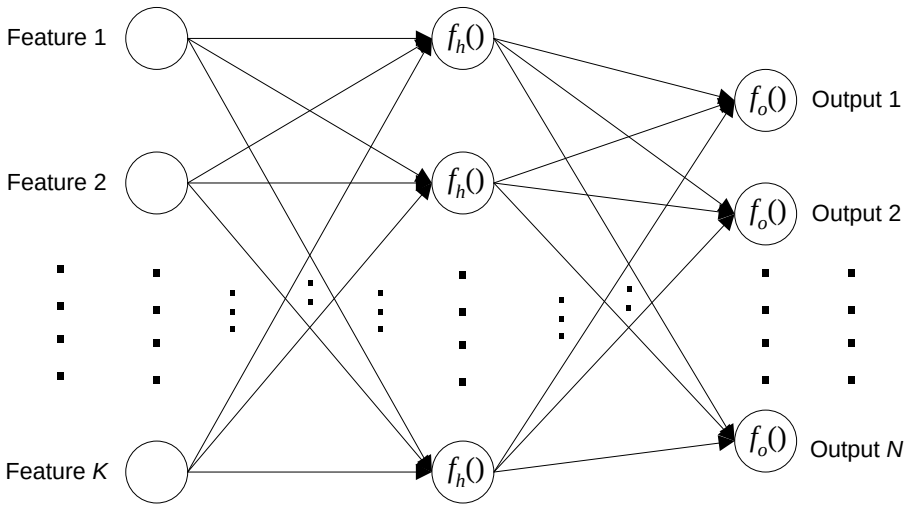


Figure 4.2 The neural network in NMMF

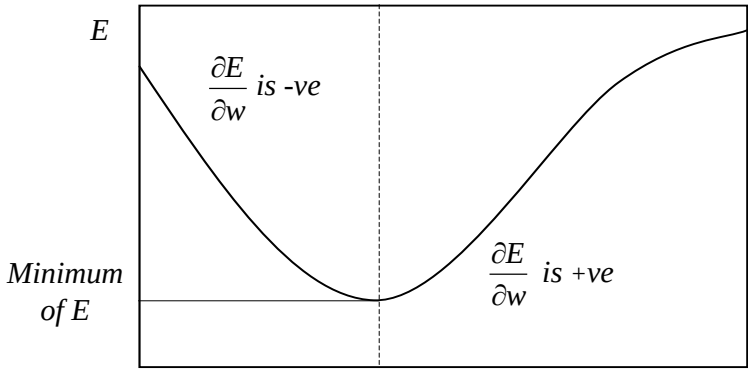


Figure 4.3 Partial derivative versus error *E*

In the following, the weights update rule for the neural network is presented. The neural network includes 3 layers: input layer, hidden layer and output layer. The back-propagation update rule is adopted to train its weights. This rule was introduced by Werbos (1974) and further enriched and popularized by Rumelhart et al. (1986). The difference between the desired output *d* and the actual output of the neural network *o* is used to direct the update of the weights. Assume that there are *K* inputs, *M* hidden neurons and *N* outputs. The training procedure adjusts the weights of the neural network

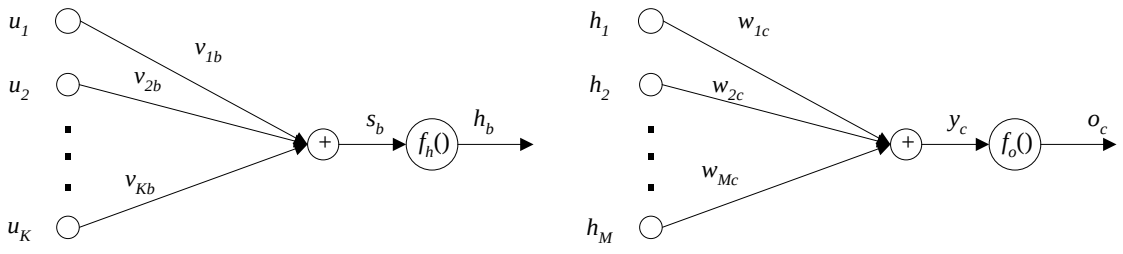
such that the mean square error $E = \frac{1}{2}\sum_{i=1}^{N}(d_i - o_i)^2$ is minimized.

The back-propagation update rule utilizes the error *E* as the feedback signal to determine the directions of adjusting weights. The derivation of update rule for a weight between hidden neurons and output neuron is elaborated firstly. Figure 4.3 is a diagram about the relationship between the partial derivative $\frac{\partial E}{\partial w}$ and error *E*. It gives the direction of modifying a weight *w*. Eq. (4.2) is the update formula for weight *w*:

$$w(new) = w(old) - \eta \frac{\partial E}{\partial w(old)} \tag{4.2}$$

where η is the learning rate which ranges between 0 and 1. Derivative value $\frac{\partial E}{\partial w}$ indicates the direction of search in all weight space.

In the following, a detailed analysis is given on the update rule for weight w_{bc} between output neuron *c* and hidden neuron *b*.



(a) (b)

Figure 4.4 Signal flow in the neural network - (a) the flow between input layer and hidden layer; (b) the flow between hidden layer and output layer

Start from the definition in Eq. (4.2), the derivation is continued by expanding $\frac{\partial E}{\partial w_{bc}}$:

$$\frac{\partial E}{\partial w_{bc}} = \frac{\partial E}{\partial y_c}\frac{\partial y_c}{\partial w_{bc}} \tag{4.3}$$

$$\frac{\partial y_c}{\partial w_{bc}} = \frac{\partial \sum_{j=1}^{M} h_j w_{jc}}{\partial w_{bc}} = \frac{\partial h_b w_{bc}}{\partial w_{bc}} + \frac{\partial \sum_{j=1(j\neq b)}^{M} h_j w_{jc}}{\partial w_{bc}} \tag{4.4}$$

where y_c is the summation input to output neuron c and $y_c = \sum_{j=1}^{M} h_j w_{jc}$; h_j is the resultant output from hidden neuron j. Since only hidden neuron b has dependence on w_{bc}, the last item on right side of Eq. (4.4) equals to 0. Eq. (4.4) is further simplified to:

$$\frac{\partial y_c}{\partial w_{bc}} = h_b \tag{4.5}$$

Substituting $E = \frac{1}{2}\sum_{i=1}^{N}(d_i - o_i)^2$ into $\frac{\partial E}{\partial y_c}$:

$$\frac{\partial E}{\partial y_c} = \frac{\partial \frac{1}{2}\sum_{i=1}^{N}(d_i - o_i)^2}{\partial y_c} = -(d_c - o_c)\frac{\partial (o_c)}{\partial y_c} = -(d_c - o_c) f_o'(\sum_{j=1}^{M} h_j w_{jc}) \tag{4.6}$$

With Eqns. (4.2), (4.3), (4.5) and (4.6), the following equation for the update rule of weight w_{bc} is derived:

$$w_{bc}(new) = w_{bc}(old) + \eta(d_c - o_c) f_o'(\sum_{j=1}^{M} h_j w_{jc}) h_b \tag{4.7}$$

Eq. (4.7) is the update rule for the weight between hidden neuron b the output neuron c, where d_c is the desired output and o_c is the actual output of output neuron c. Similarly the update rule for the weight between input a and hidden neuron b can be derived. As shown in Figure 4.4, to distinguish the weights between hidden and output layer, the symbol v is

used instead. Expanding $\frac{\partial E}{\partial v_{ab}}$:

$$\frac{\partial E}{\partial v_{ab}} = \frac{\partial E}{\partial s_b}\frac{\partial s_b}{\partial v_{ab}} = \frac{\partial E}{\partial h_b}\frac{\partial h_b}{\partial s_b}\frac{\partial s_b}{\partial v_{ab}} \tag{4.8}$$

Where s_b is the summation signal input to hidden neuron b and $s_b = \sum_{i=1}^{K} u_i v_{ib}$, u_i is the signal from input i; h_b is the output of hidden neuron b and $h_b = f_h(s_b)$. It is obvious that:

$$\frac{\partial h_b}{\partial s_{ab}} = f_h'(s_b) \tag{4.9}$$

$$\frac{\partial s_b}{\partial v_{ab}} = u_a \tag{4.10}$$

Subsequently, the term $\frac{\partial E}{\partial h_b}$ is further expanded:

$$\frac{\partial E}{\partial h_b} = \frac{\partial}{\partial h_b}\{\frac{1}{2}\sum_{i=1}^{N}[d_i - f_o(y_i)]^2\} = -\sum_{i=1}^{N}(d_i - o_i) f_o'(y_i)\frac{\partial y_i}{\partial h_b} \tag{4.11}$$

Since $y_i = \sum_{j=1}^{M} h_j w_{ji}$, Eq. (4.11) can be written as:

$$\frac{\partial E}{\partial h_b} = -\sum_{i=1}^{N}(d_i - o_i) f_o'(y_i) w_{bi} \tag{4.12}$$

With Eqns. (4.2), (4.9), (4.10) and (4.12), the equation for the update rule of weight v_{ab} is derived:

$$v_{ab}(new) = v_{ab}(old) + \eta u_a f_h'(s_b)\sum_{i=1}^{N}(d_i - o_i) f_o'(y_i) w_{bi} \tag{4.13}$$

In NMMF, identity and *tanh* functions form the activation functions of the output and

hidden neurons respectively:

$$f_o(I_o) = I_o; \quad f_h(I_h) = \tanh(I_h) \tag{4.14}$$

The corresponding derivatives are:

$$f_o'(I_o) = 1; \quad f_h'(I_h) = 1 - \tanh^2(I_h) \tag{4.15}$$

Substituting the two derivatives into Eq. (4.15) into Eqns. (4.7) and (4.13) respectively, the two weights update formulas are defined as follow:

$$v_{ab}(new) = v_{ab}(old) + \eta_{IH}(1 - h_b^{\,2})u_a \sum_{i=1}^{N}(d_i - o_i)w_{bi} \tag{4.16}$$

$$w_{bc}(new) = w_{bc}(old) + \eta_{HO}(d_c - o_c)h_b \tag{4.17}$$

where *a* denotes the neuron index of the input layer, *b* is the neuron index of the hidden layer while *c* denotes the neuron index of the output layer. Additionally, *N* denotes the number of outputs, η_{IH} and η_{HO} are the learning rates for the weights in hidden and output layers respectively, u_a is the signal from input *a* and h_b is the resultant output of hidden neuron *b* (refer to Figure 4.4).

With the weights update formulas defined by Eqns. (4.16) and (4.17) and the training data produced by the training data generation system with small-scale problems, the neural network in NMMF can be trained with relatively simple steps and acceptable computational time cost.

The training data generation procedures and neural network utilize the experience collected from solving relatively small-scale combinatorial optimization problems to help in the construction of appropriate meta-memes for previously unsolved problems. This is particularly advantageous for large-scale problems since generating optimized control parameters from scratch is not practical due to the unmanageable computational time cost.

4.3 Dominance and Sparsity

In (Vollmann and Buffa 1966), the flow dominance (*fd*), as shown in Eq. (4.18), was introduced as a means to characterize the QAP:

$$fd(c) = \frac{a}{b};\ a = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ij}-b)^2}{n^2}};\quad b = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}}{n^2} \tag{4.18}$$

where c_{ij} is the entry number in position (i, j) in the flow matrix and n is the problem size. Similar to the flow dominance, the distance dominance for a distance matrix is also defined. Another coefficient is the sparsity which is calculated as in Eq. (4.19):

$$sp = \frac{n_0}{n^2} \tag{4.19}$$

where n_0 is the total number of zero entries in a matrix, n is the size of the matrix. The sparsity gives the proportion of zero entries with reference to the total number of entries.

Real-life QAPs usually have large sparsity values. In Table 4.1, dominance and sparsity values for several selected benchmarks are presented. The number tagged to a benchmark's name indicates the size of that problem.

Table 4.1 Dominance and sparsity values for some QAP benchmarks

Benchmarks	Problem Size	Flow Dominance	Distance Dominance	Flow Sparsity	Distance Sparsity
bur26a	26	2.749	0.151	0.223	0.000
bur26b	26	2.749	0.159	0.223	0.000
bur26c	26	2.284	0.151	0.257	0.000
els19	19	5.310	0.521	0.690	0.053
kra30a	30	1.500	0.492	0.033	0.633
kra30b	30	1.500	0.500	0.633	0.033
ste36a	36	4.003	0.556	0.735	0.028
ste36b	36	4.003	1.008	0.735	0.028
ste36c	36	4.003	0.559	0.735	0.028
tai40b	40	3.172	0.667	0.528	0.025
tai50b	50	3.139	0.734	0.568	0.020
tai60b	60	3.178	0.768	0.564	0.017
tai80b	80	3.232	0.640	0.565	0.013
tai100b	100	3.213	0.804	0.562	0.010
tai40a	40	0.602	0.631	0.034	0.039
tai50a	50	0.622	0.607	0.024	0.033
tai60a	60	0.609	0.614	0.027	0.024
tai80a	80	0.604	0.592	0.021	0.023
tai100a	100	0.603	0.593	0.021	0.018
tho40	40	1.555	0.532	0.610	0.025
tho150	150	1.472	0.515	0.579	0.007
wil50	50	0.667	0.542	0.121	0.020
wil100	100	0.645	0.508	0.108	0.010
sko81	80	1.066	0.509	0.307	0.012
sko90	90	1.075	0.509	0.316	0.011
sko100a	100	1.066	0.508	0.314	0.010
sko100b	100	1.084	0.508	0.317	0.010
sko100c	100	1.081	0.508	0.326	0.010

Table 4.1 shows that real-life and real-life like problems present the largest flow dominances among all the QAP benchmarks. Higher value of flow dominance indicates the tendency for fewer facilities contributing to the overall flow. The relatively small flow and distance dominances indicate relatively even flow and distance between all facilities and locations, and result in more unstructured and random problems, such as the

taixxa type. The difference between dominance values of different groups of QAPs is conspicuous, and there is only slight difference between dominance values of QAPs in the same group. Experience has shown that the performance of a search method is relatively consistent for the QAPs within the same group. This fact provides theoretical evidence for the usage of dominance values as coefficients in generating different appropriate search mechanisms for different QAPs. Similar to the dominance values, the sparsity values are chosen as means to characterize QAP. In summary, there are four coefficients used in NMMF as feature numbers in producing appropriate mechanisms for different QAPs: flow dominance (*fd*), distance dominance (*dd*), flow sparsity (*fsp*) and distance sparsity (*dsp*).

4.4 Neural Meta-Memes Framework for QAP

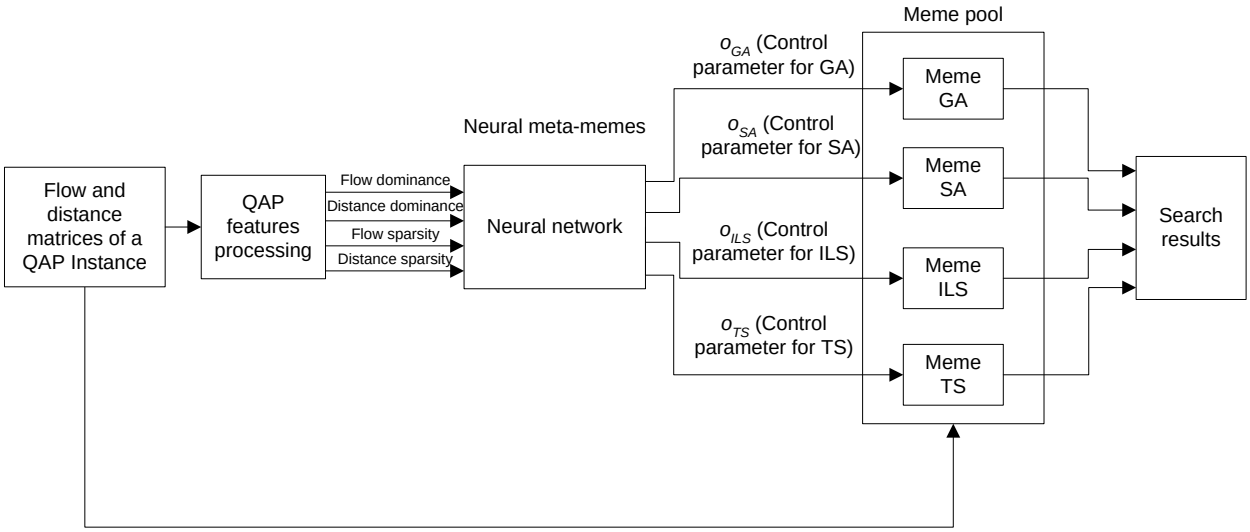


Figure 4.5 Neural meta-memes framework for QAP

In this section, details about the NMMF specifically designed for QAP are elaborated. In Figure 4.5, the neural meta-memes framework for solving QAP is depicted. The search methods which are modeled as memes in NMMF are GA, SA, ILS and TS, details about these methods were discussed in Chapter 2.

4.4.1 Generation of Training Data

In Section 4.2 the general procedures of generating training data for NMMF have been discussed. In this section the procedures of generating optimized control parameters with the four memes are presented. The parameters that characterize a mechanism are o_{GA}, o_{SA}, o_{ILS}, o_{TS}, the continuous iterations without improvement allowed for GA, SA, ILS and TS

respectively. At the beginning of the learning process, the parameters are initialized with a set of values which have been tested and achieved acceptable results for most of the QAP benchmarks. After generations, the parameters which produce the best result are the optimized control parameters specifically for that problem under test. Since the process is a form of greedy learning, the expected computational time cost in this part will be predictably long for large-scale combinatorial optimization problems. In the test here, some relatively small-scale QAP benchmarks are used to generate the training data. The training data consist of features of the selected small-scale problems and their associated optimized control parameters.

It is noted that the NMMF allows for the incorporation of useful information based on earlier problem-solving sessions and expert knowledge. Based on experience, the four optimization methods or memes, i.e., GA, SA, ILS and TS, synergize well when the range of computational budget in terms of iterations in stagnation is configured in the ranges of [0, 6] for o_{GA}, [1, 5] for o_{SA}, [0, 3] for o_{ILS} and [99, 111] for o_{TS}. This forms the initial settings used in the neural meta-meme training process.

4.4.2 Neural Network

The neural network used in NMMF for QAP is responsible for generating control parameters for the four memes. In Section 4.3 the QAP features have been discussed, in other words, the four inputs for the neural network: flow dominance, distance dominance, flow sparsity and distance sparsity. The inputs of the neural network are designed to be

adapted with the QAP just for evaluation purpose. To adapt with other types of combinatorial optimization problems, the inputs can be modified according to the characteristics of the corresponding problems.

As discussed in Section 4.2, the direct outputs from the neural network are the four normalized values o'_{GA}, o'_{SA}, o'_{ILS} and o'_{TS}, the following equation is used to calculate the actual parameters:

$$o_{xx} = o'_{xx} \times o_{xx\max} \quad (4.20)$$

where o'_{xx} refers to any one of the four direct outputs from the neural network; o_{xx}, refers to the corresponding actual parameter (rounded down to an integer); [o_{xxmin}, o_{xxmax}] defines the upper and lower bounds of o_{xx}.

Figure 4.6 illustrates the workflow for the training of the neural network using the back-propagation update rule described in Section 4.2. With the literature survey, the learning rates η_{IH} and η_{HO} are configured as moderate values of 0.7 and 0.07, respectively. Here $\mathbf{i}^{\mathbf{T}}$ and $\mathbf{d}^{\mathbf{T}}$ are training patterns which contain the input parameters [*fd*, *dd*, *fs*, *ds*] and the corresponding designed output parameters [d_{GA}, d_{SA}, d_{ILS}, d_{TS}], respectively. Here [*fd*, *dd*, *fs*, *ds*] refer to dominance and sparsity which have been discussed in Section 4.3. [d_{GA}, d_{SA}, d_{ILS}, d_{TS}] refer to the control parameters generated with the procedures described in Section 4.4.1. They define the continuous iterations without improvement allowed for GA, SA, ILS and TS. The parameter n_{gen} is the count number which records the number of generations of the training. It is initialized to 0 and increases by 1 after every

generation of the training process. Parameter n_{max} defines the maximum number of generations allowed for the training. When n_{gen} is equal to or greater than n_{max}, training will be terminated. Parameter n_{pat} refers to the total number of training patterns available in $\mathbf{i}^{\mathbf{T}}$ and $\mathbf{d}^{\mathbf{T}}$. Parameter $n_{trained}$ refers to the number of patterns which have already been selected and trained with the neural network. It is initialized to 0 in the beginning of each generation and increases by 1 after a pattern is selected for the training. Parameter $n_{trained}$ is used to set a limit for the number of patterns selected in every generation (here the limit is n_{pat}). The $\mathbf{o}^{\mathbf{T}}$ shown in Figure 4.6 contains the actual outputs from the neural network [o_{GA}, o_{SA}, o_{ILS}, o_{TS}]. The difference between $\mathbf{d}^{\mathbf{T}}$ and $\mathbf{o}^{\mathbf{T}}$ is the error signal used to direct the update of the weights. Other parameters in the two update formulas have been discussed in details in Section 4.2.2.

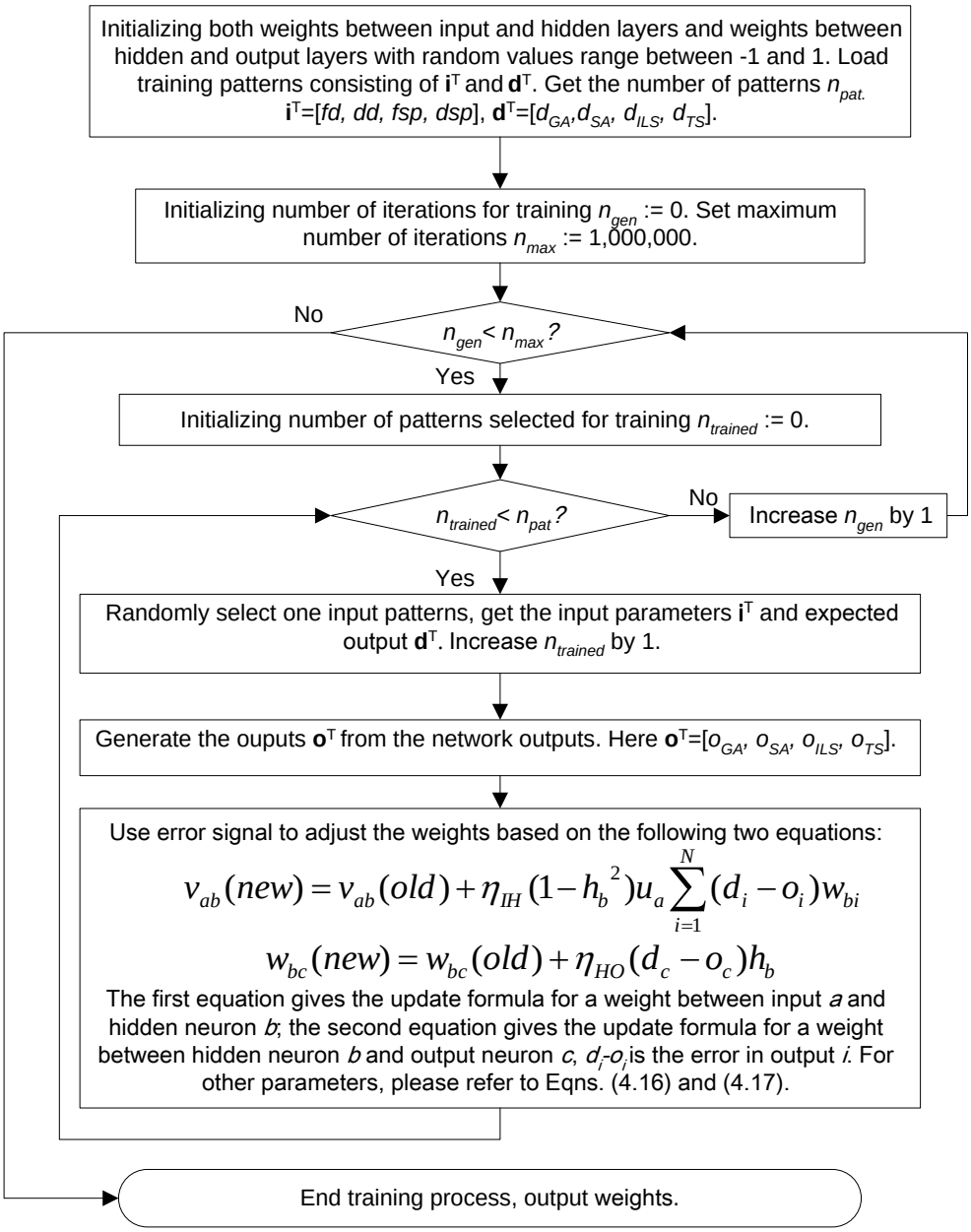


Figure 4.6 Workflow of training the neural network in NMMF

4.5 Experimental Results and Analysis

4.5.1 Experimental Results

Table 4.2 Dominance, sparsity values and the corresponding optimized control parameters for QAP benchmarks as training dataset

Benchmark	Flow dominance (fd)	Distance dominance (dd)	Flow sparsity (fsp)	Distance sparsity (dsp)	o_{GA}	o_{SA}	o_{ILS}	o_{TS}
lipa30a	0.429	0.322	0.0333	0.0656	0	4	2	101
lipa30b	0.429	0.641	0.0333	0.0678	1	3	2	100
lipa40a	0.421	0.277	0.0250	0.0494	0	3	3	100
lipa40b	0.421	0.642	0.0250	0.0519	1	5	2	100
lipa50a	0.412	0.247	0.0200	0.0396	0	4	1	100
lipa50b	0.412	0.618	0.0200	0.0412	1	4	2	100
lipa60a	0.421	0.226	0.0167	0.0331	0	3	3	99
lipa60b	0.421	0.605	0.0167	0.0311	1	5	2	102
lipa70a	0.419	0.209	0.0143	0.0284	0	5	1	100
lipa70b	0.419	0.603	0.0143	0.0302	1	4	2	101
sko42	1.085	0.520	0.3163	0.0238	1	4	0	108
sko49	1.094	0.515	0.3244	0.0204	1	3	1	105
sko56	1.105	0.515	0.3233	0.0179	1	3	0	111
sko64	1.084	0.512	0.3232	0.0156	1	3	0	107
tai17a	0.690	0.643	0.0727	0.0796	5	3	0	101
tai20a	0.649	0.670	0.0650	0.0650	5	1	1	101
tai25a	0.643	0.618	0.0560	0.0432	5	4	0	100
tai30a	0.632	0.580	0.0467	0.0400	5	3	1	99
tai35a	0.616	0.616	0.0384	0.0433	6	4	0	101
tai20b	3.332	1.283	0.4600	0.0500	1	4	0	100
tai25b	3.104	0.870	0.4272	0.0400	2	3	1	99
tai30b	3.239	0.852	0.4656	0.0356	1	3	2	99
tai35b	3.096	0.787	0.5527	0.0286	2	5	1	104
tai40b	3.172	0.667	0.5275	0.0250	0	3	0	99
tai50b	3.139	0.734	0.5676	0.0200	1	4	1	100
tai60b	3.178	0.768	0.5644	0.0167	1	3	2	100
tho30	1.379	0.593	0.5178	0.0333	1	4	2	100
tho40	1.555	0.532	0.6100	0.0250	1	4	1	105
wil50	0.667	0.542	0.121	0.020	2	4	4	99

The 29 QAP benchmarks in Table 4.2 are chosen to generate training data with the training data generation system. These benchmarks are selected because they all belong to groups which also contain larger size instances which are relatively difficult to solve,

and these benchmarks themselves can be solved to optimality with relatively short computational time. The neural network trained based on the training dataset is endowed with the capability to associate the characteristics of the QAPs based on the problem features with suitably configured mechanisms for larger scale or more complex problems in their corresponding groups.

With the training data generation system, optimized control parameters were generated for the 29 benchmarks. The parameters are presented in columns o_{GA}, o_{SA}, o_{ILS} and o_{TS} of Table 4.2. These generated parameters together with the corresponding dominance and sparsity values form the training dataset for the neural network.

With the training data provided in Table 4.2, the neural network responsible for producing the optimized parameters for different problems was trained for 1,000,000 generations in about 15 minutes. With the optimized control parameters produced by the neural network, 106 QAP benchmarks which form the unsolved test problems set plus the 29 problems given in Table 4.2 were tested with NMMF where 20 separate runs were carried out for each problem. For each run, a computational time budget of around 1000 seconds is allocated.

The parametric settings of the four optimization methods are as follow:

--*Tabu Search*: tabu length = problem size.

--*Simulated Annealing*: starting temperature = 30.0; temperature annealing factor α=

0.9998; maximum number of neighbors searched in one generation = 200.

--*Iterated Local Search*: worse solution acceptance criterion c = 0.0; minimum perturbation size k_{min} = 3; maximum perturbation size k_{max} = min(0.9*problem size, 15).

--*Genetic Algorithm*: crossover rate = 0.5, mutation rate = 0.05; mutation range = 0.01.

Another common parameter is the population size which is set to 60.

The computing platform in the simulation study is an Intel Core 2 DUO 2.66Ghz CPU with 2GB memory in Windows XP operating system.

In Table 4.3, the results for the 42 QAP benchmarks which were solved to optimality for all 20 runs by NMMF are presented. Here those instances which the average computational time costs were below or equal to 0.5 seconds are not included; they are *bur26x*, *hadxx*, *esc32x* (exclude *esc32a*), *esc64a*, *lipa20b* to *lipa50b*, *lipa20a*, *lipa30a*, *nugxx*, *rouxx*, *scrxx*, *tho30*, *tai10a* to *tai20a* and *tai10b* to *tai25b*, etc.

In Table 4.4 the results of the 15 benchmarks which were not solved to optimality for all 20 runs by NMMF are listed. *Optimum solution* refers to the best known or optimum solution value. *Average cost* refers to the average solution value. *Mechanism* refers to the control parameters generated with NMMF for every benchmark; it is presented in the form of (o_{GA}, o_{SA}, o_{ILS}, o_{TS}). *Min time* refers to the shortest CPU time elapsed when the

best solution found occurs. *Ave time* refers to the average CPU time cost for all runs.

Table 4.3 Results on QAP benchmarks which were solved to optimality for all 20 runs by NMMF

Benchmark	Mechanism	Optimum solution	Average performance	Min time(s)	Ave time(s)	Max time(s)
chr20a	(6,3,2,101)	2192	2192	0.06	1.30	5.30
chr20b	(6,3,2,101)	2298	2298	0.17	8.00	31.38
chr20c	(6,3,2,101)	14142	14142	0.01	0.78	5.19
chr22a	(6,3,2,101)	6156	6156	0.09	0.98	2.08
chr22b	(6,3,2,101)	6194	6194	0.52	1.81	4.25
chr25a	(6,3,2,101)	3796	3796	1.01	7.47	20.69
esc32a	(5,3,3,101)	130	130	0.16	3.06	13.94
esc128	(1,4,2,101)	64	64	0.66	1.48	1.72
kra30a	(0,3,3,101)	88900	88900	0.25	0.70	1.76
kra30b	(0,3,0,99)	91420	91420	0.20	0.80	2.20
kra32	(0,3,3,101)	88700	88700	0.20	0.70	1.72
ste36a	(1,3,0,111)	9526	9526	0.70	3.91	11.98
ste36b	(1,3,0,101)	15852	15852	0.27	1.38	2.94
ste36c	(1,3,0,111)	8239110	8239110	1.31	2.64	4.13
lipa60b	(0,4,2,101)	2520135	2520135	0.75	1.43	2.95
lipa70b	(1,4,2,101)	4603200	4603200	1.09	2.39	3.70
lipa80b	(0,4,2,101)	7763962	7763962	1.92	4.67	8.19
lipa90b	(0,4,2,101)	12490441	12490441	3.58	9.02	15.94
lipa40a	(0,3,2,100)	31538	31538	0.45	1.01	2.16
lipa50a	(0,3,2,100)	62093	62093	2.19	4.01	6.53
lipa60a	(0,3,2,100)	107218	107218	5.03	12.07	20.70
lipa70a	(0,4,1,100)	169755	169755	14.22	31.94	101.44
lipa80a	(0,5,0,101)	253195	253195	82.42	141.39	222.41
lipa90a	(0,5,0,102)	360630	360630	56.22	285.71	587.11
sko42	(0,3,0,106)	15812	15812	1.38	2.43	4.28
sko49	(1,3,0,107)	23386	23386	2.80	18.99	50.13
sko56	(1,3,0,108)	34458	34458	6.17	17.93	43.13
sko64	(1,3,0,107)	48498	48498	6.08	32.26	64.47
sko72	(1,3,0,107)	66256	66256	37.00	97.19	201.00
sko81	(1,3,0,107)	90998	90998	40.02	276.61	978.51
sko100b	(1,3,0,108)	153890	153890	74.16	276.87	744.86
tai25a	(5,3,0,101)	1167256	1167256	0.23	2.36	8.84
tai30a	(5,3,0,101)	1818146	1818146	0.25	3.24	10.72
tai35a	(5,3,0,101)	2422002	2422002	1.48	18.98	104.45
tai30b	(1,3,0,99)	637117113	637117113	0.19	1.16	2.66
tai35b	(1,3,0,102)	283315445	283315445	0.58	2.50	5.08
tai40b	(1,3,0,100)	637250948	637250948	0.64	1.94	3.25
tai50b	(1,3,0,103)	458821517	458821517	9.86	17.25	29.50
tai60b	(1,3,0,102)	608215054	608215054	9.52	28.80	45.81
tai80b	(1,3,0,106)	818415043	818415043	64.77	150.06	649.84
tho40	(1,3,0,106)	240516	240516	1.42	18.34	71.80
wil50	(2,3,0,101)	48816	48816	1.88	42.37	256.69

Max time refers to the longest CPU time cost in all runs. *Worst gap* refers to the percentage difference between the worst solution found and the optimum solution. Other

terms were explained in Section 3.4.2.

Table 4.4 Results on QAP benchmarks which were not solved to optimality for all 20 runs by NMMF

Benchmark	Mechanism	Optimum solution	Success rate(%)	Best gap(%)	Average gap(%)	Worst gap(%)	Ave time(s)	Max time(s)
sko90	(1,3,0,108)	115534	90	0	0.0038	0.038	281.44	862.13
sko100a	(1,3,0,108)	152002	75	0	0.0088	0.058	309.73	849.69
sko100c	(1,3,0,108)	147862	95	0	0.000068	0.0014	406.94	858.81
sko100d	(1,3,0,108)	149576	75	0	0.0072	0.043	391.51	967.75
sko100e	(1,3,0,109)	149150	90	0	0.00047	0.0054	492.28	861.34
sko100f	(1,3,0,108)	149036	80	0	0.0048	0.075	420.21	958.13
tai40a	(5,3,0,101)	3139370	5	0	0.088	0.15	285.85	561.55
tai50a	(5,3,0,101)	4938796	5	0	0.35	0.58	378.83	831.50
tai60a	(5,3,0,101)	7205962	0	0.16	0.33	0.54	513.96	949.39
tai80a	(5,3,0,101)	13511780	0	0.32	0.47	0.68	546.86	925.06
tai100a	(5,4,0,100)	21052466	0	0.26	0.43	0.68	644.63	991.33
tai100b	(1,3,0,101)	1185996137	80	0	0.025	0.14	261.15	625.75
tai150b	(1,3,0,111)	498896643	0	0.0016	0.33	0.53	976.53	1003.65
tho150	(1,3,0,103)	8133398	5	0	0.039	0.069	893.01	1002.28
wil100	(2,3,0,101)	273038	25	0	0.0036	0.039	259.99	761.66

4.5.2 Comparison

Table 4.5 Comparison of test results for NMMF and ES-ILS (Stützle 2006)

Benchmark	Optimum Solution	Neural Meta-Memes Framework (NMMF)			Evolution Strategy-Iterated Local Search (ES-ILS)		
		Success rate(%)	Best gap(%)	Average gap(%)	Success rate(%)	Best gap(%)	Average gap(%)
tai40a	3139370	**5**	**0**	**0.088**	0	0.074	0.28
tai50a	4941410	**15**	**-0.053**	**0.30**	0	0.34	0.61
tai60a	7205962	0	**0.16**	**0.33**	0	0.68	0.82
tai80a	13540420	0	**0.11**	**0.26**	0	0.46	0.62
tai100a	21123042	**25**	**-0.077**	**0.097**	0	0.54	0.69
sko72	66256	**100**	**0**	**0**	88	**0**	0.0012
sko81	90998	**100**	**0**	**0**	52	**0**	0.0074
sko90	115534	**90**	**0**	**0.0038**	40	**0**	0.0057
sko100a	152002	**75**	**0**	**0.0088**	30	**0**	0.012
sko100b	153890	**100**	**0**	**0**	50	**0**	0.0068
sko100c	147862	**95**	**0**	**0.000068**	60	**0**	0.0023
sko100d	149576	**75**	**0**	**0.0072**	0	0.0013	0.021
sko100e	149150	**90**	**0**	**0.00047**	70	**0**	0.0013
sko100f	149036	**80**	**0**	**0.0048**	0	0.023	0.037
wil100	273038	**25**	**0**	**0.0036**	10	**0**	0.0041
tho150	8133398	**5**	**0**	**0.039**	0	0.041	0.068
tai100b	1185996137	80	**0**	0.025	**100**	**0**	**0**
tai150b	498896643	0	0.0016	0.33	0	**0.0003**	**0.095**

In Table 4.5, the results of 18 instances by both NMMF and the evolutionary strategy iterative local search method (ES-ILS) presented in (Stützle 2006) are listed. The better success rate, best gap and average gap are highlighted in bold for each benchmark. The optimum solution values for *tai50a*, *tai80a* and *tai100a* in Table 4.5 are different from the corresponding values in Table 4.4. This is because in Table 4.4, the up to date optimum solutions with lower costs reported in (Burkard et al. 1997) were adopted, but in (Stützle 2006) older versions of the three optimum solutions were adopted. To maintain consistency, the best gaps and the average gaps on these three instances are recalculated for NMMF based on the optimum solutions adopted in (Stützle 2006). The best gaps for *tai50a* and *tai100a* given by NMMF are negative values because the best solutions found by NMMF are better than the optimum solution values adopted in (Stützle 2006). The comparison in Table 4.5 shows that out of the 18 benchmarks, NMMF outperforms ES-ILS for 16 cases with only two exceptions, *tai100b* and *tai150b*. For these 16 cases, NMMF presents better average and best solution values than ES-ILS. For *tai40a*, *tai50a*, *tai60a*, *tai80a* and *tai100a*, NMMF gives average solution gaps of 0.088%, 0.30%, 0.33%, 0.26% and 0.097, which are much lower than the values of 0.28%, 0.61%, 0.82%, 0.62% and 0.69% by ES-ILS respectively. The comparison also shows that the average gaps for *tai50a*, *tai60a*, *tai80a* and *tai100a* by NMMF are even less than the best gaps for the same instances by ES-ILS; this suggests that NMMF substantially improves the search quality for *taixxa* instances compared to ES-ILS. NMMF also presents much higher success rates on most of the 16 benchmarks. For

example, NMMF gives success rate of 100% on *sko72, sko81, sko100b* while ES-ILS only gives the success rates of 88%, 52% and 50% for the three instances. With the optimum solution values adopted in (Stützle 2006), NMMF managed to solve *tai40a*, *tai50a*, *tai100a*, *sko100d*, *sko100f* and *tho150* to optimality with corresponding success rates of 5%, 15%, 25%, 65%, 80% and 5%, while ES-ILS failed to solve any of the 6 instances to optimality in all its trails.

It is noted that the training time of the neural network is not considered for the comparison of the test results. The reason is that the training time cost becomes trivial compared to the total computational time budget allocated to all trials for all QAP instances. The training of the neural network is only performed once during the entire test. Also the training of the neural network and the testing of all 135 QAP instances can be considered as two different phases. The current computational time budget allocated to the training phase is about 15 minutes. From the observation during the experiment, this value can even be much smaller without affecting the performance. For the testing phase, the total allocated computational time budget can be calculated as $1000 \times 20 \times 135$ (1000 seconds for each run and 20 runs for each of 135 instances), which is 2,700,000 seconds or 45,000 minutes. The time budget allocated for the testing phase is much larger than the budget allocated for the training phase. It can also be analyzed from a different aspect: if the training time is merged into each run, it equals to $(15 \times 60)/(20 \times 135)$, which is 0.33 seconds. This value is insignificant compared to the actual testing time for each run which averages around hundreds of seconds (the allocated time budget is 1000 seconds).

This is the reason why the training time is not taken into consideration for the comparison of the test results.

From Tables 4.3 and 4.4, it is noticed that NMMF tends to generate similar or even the same optimized mechanisms for problem instances within the same group, for example, mechanism (1,3,0,108) for *skoxx* instances, (5,3,0,101) for *taixxa* instances and (0,3,2,100) for *lipaxxa* instances. The reason is NMMF generates a mechanism based on the four feature values of a QAP instance. It is clear that the instances within the same group have very close dominance and sparsity values.

Table 4.6 Test results for cases with non-optimized mechanisms

Benchmark	Mechanism	Optimum solution	Success rate(%)	Best gap(%)	Average gap(%)	Worst gap(%)	Ave time(s)	Max time(s)
lipa90a	(5,3,0,101)	360630	80	0	0.085	0.44	485.54	866.70
sko81	(5,3,0,101)	90998	90	0	0.0011	0.011	252.86	943.73
sko90	(5,3,0,101)	115534	70	0	0.020	0.095	207.29	500.34
sko100a	(5,3,0,101)	152002	55	0	0.019	0.058	370.82	987.67
sko100b	(5,3,0,101)	153890	85	0	0.0059	0.040	296.77	969.47
sko100c	(5,3,0,101)	147862	70	0	0.00041	0.0014	440.53	781.42
sko100d	(5,3,0,101)	149576	50	0	0.019	0.066	456.73	973.88
sko100e	(5,3,0,101)	149150	65	0	0.0014	0.0040	358.42	999.17
sko100f	(5,3,0,101)	149036	50	0	0.017	0.076	274.31	764.76
tai40a	(1,3,0,108)	3139370	5	0	0.086	0.33	393.92	968.96
tai50a	(1,3,0,108)	4938796	0	0.29	0.48	0.64	439.87	932.02
tai60a	(1,3,0,108)	7205962	0	0.42	0.63	0.76	507.95	981.83
tai80a	(1,3,0,108)	13511780	0	0.52	0.74	0.91	603.59	1000.31
tai100a	(1,3,0,108)	21052466	0	0.50	0.78	0.91	605.59	969.80

To validate that the generated mechanisms for different groups of instances are optimized, 14 QAP instances from *lipaxxa*, *skoxx* and *taixxa* groups were tested with “non-optimized” mechanisms. Here mechanism (5,3,0,101) was tested with *skoxx* and *lipaxxa* instances; mechanism (1,3,0,108) was tested with *taixxa* instances. The experimental results are presented in Table 4.6. Table 4.7 summarizes the comparison between results generated

from “optimized” and “non-optimized” control mechanisms. For 13 out of the 14 cases, the optimized mechanisms produced convincingly better average solution values and higher success rates. For example, optimized mechanisms can solve *lipa90a*, *sko80a* and *sko100b* to optimality for all runs, while the non-optimized mechanisms failed to find their optimum solutions in some trials with success rates of 80%, 90% and 85% respectively. Compared to the results for *taixxa* group by the non-optimized mechanism, the optimized mechanisms produced much lower average solution costs except for *tai40a* which the non-optimized mechanism presented a slightly better average gap.

Table 4.7 Comparison of test results by optimized and non-optimized mechanisms

Benchmark	Optimum Solution	Results produced by optimized mechanisms			Results produced by non-optimized mechanisms		
		Success rate(%)	Best gap(%)	Average gap(%)	Success rate(%)	Best gap(%)	Average gap(%)
lipa90a	360630	**100**	**0**	**0**	80	**0**	0.085
sko81	90998	**100**	**0**	**0**	90	**0**	0.0011
sko90	115534	**90**	**0**	**0.0038**	70	**0**	0.020
sko100a	152002	**75**	**0**	**0.0088**	55	**0**	0.019
sko100b	153890	**100**	**0**	**0**	85	**0**	0.0059
sko100c	147862	**95**	**0**	**0.000068**	70	**0**	0.00041
sko100d	149576	**75**	**0**	**0.0072**	50	**0**	0.019
sko100e	149150	**90**	**0**	**0.00047**	65	**0**	0.0014
sko100f	149036	**80**	**0**	**0.0048**	50	**0**	0.017
tai40a	3139370	**5**	**0**	0.088	**5**	**0**	**0.086**
tai50a	4938796	**5**	**0**	**0.35**	0	0.29	0.48
tai60a	7205962	**0**	**0.16**	**0.33**	0	0.42	0.63
tai80a	13511780	**0**	**0.32**	**0.47**	0	0.52	0.74
tai100a	21052466	**0**	**0.26**	**0.43**	0	0.50	0.78

With the comparison results in Table 4.7, it is reasonable to conclude with high level of certainty that the mechanism generated for every problem by NMMF can find the best search results compared to other mechanisms.

4.5.3 Simulations on New QAP Benchmarks

The NMMF was tested further with a group of new QAP instances provided by Drezner (Drezner et al. 2005). These instances were claimed to be especially difficult for metaheuristic methods. These instances were also tested with four memes independently for comparison purpose. All the configurations are the same as for previous simulations.

Table 4.8 Comparison of test results on *drexx* instances by different methods

Benchmark	Method	Success rate(%)	Best solution	Best gap(%)	Average solution	Average gap(%)	Ave time(s)	Max time(s)
Dre30 508	GA	0	1490.0	193.31	1740.5	242.62	0.01	0.03
	ILS	0	1502.0	195.67	1622.3	219.35	423.82	895.11
	SA	0	560.0	10.24	669.0	31.69	427.33	991.27
	TS	**100**	**508.0**	**0**	**508.0**	**0**	6.69	21.16
	NMMF	**100**	**508.0**	**0**	**508.0**	**0**	4.40	11.64
Dre42 764	GA	0	2554.0	234.29	2887.3	277.92	0.04	0.20
	ILS	0	2670.0	249.48	2848.7	272.87	594.23	990.30
	SA	0	1088.0	42.41	1125.0	47.25	428.01	920.24
	TS	**100**	**764.0**	**0**	**764.0**	**0**	76.07	191.86
	NMMF	**100**	**764.0**	**0**	**764.0**	**0**	32.72	103.17
Dre56 1086	GA	0	4104.0	277.90	4488.6	313.31	0.05	0.08
	ILS	0	4354.0	300.92	4506.7	314.98	628.11	943.17
	SA	0	1738.0	60.04	1813.1	66.95	461.42	887.61
	TS	30	**1086.0**	**0**	1248.1	14.93	430.68	918.63
	NMMF	**50**	**1086.0**	**0**	**1192.9**	**9.84**	259.69	976.69
Dre72 1452	GA	0	5852.0	303.03	6246.1	330.17	0.09	0.16
	ILS	0	6312.0	334.71	6437.7	343.37	566.31	918.80
	SA	0	2400.0	65.29	2536.3	74.68	620.36	994.80
	TS	0	1898.0	30.72	2014.1	38.71	551.64	937.20
	NMMF	**65**	**1452.0**	**0**	**1534.0**	**5.65**	417.01	967.03
Dre90 1838	GA	0	7846.0	326.88	8415.3	357.85	0.11	0.17
	ILS	0	8266.0	349.73	8534.1	364.31	444.04	964.20
	SA	0	3344.0	81.94	3450.9	87.75	594.98	953.22
	TS	0	2074.0	12.84	2530.0	37.65	846.08	996.39
	NMMF	0	**2016.0**	**9.68**	**2330.2**	**26.78**	716.01	999.76

The comparison results are shown in Table 4.8. In the *benchmark* column, numbers under names are their corresponding optimum solution values. For each benchmark, the best performance among all the five search methods is highlighted in bold. The average solution values for the 5 instances by the five methods are also plotted in Figures 4.7 to

4.9. The results in Table 4.8 and Figures 4.7 to 4.9 indicate that NMMF outperforms the four memes independently in terms of both success rates and solution costs. For example, NMMF solved *dre30* and *dre42* to optimality for all 20 trials and presented success rates of 50% and 65% for *dre50* and *dre72* while the four memes independently all failed to solve *dre72*. Only tabu search produced average solution costs which are close to NMMF and even was able to solve *dre30* and *dre42* in all 20 runs, but the average computational time costs for these two are longer than NMMF; average computational time costs by tabu search for the two cases are 6.69 and 76.07 seconds while NMMF only takes 4.40 and 32.72 seconds. Except for the NMMF, tabu search produced the best search performance among all four methods with much better average solution costs. This is an intuitive validation of the fact that it is reasonable for tabu search to be allocated the largest proportion of the search effort for all cases tested. The experiments on these 5 benchmarks further prove the robustness and effectiveness of NMMF. The comparison between the computational time costs by tabu search and NMMF further shows that not only the solution quality is improved with NMMF compared with the memes independently; the search efficiency is also enhanced evidently.

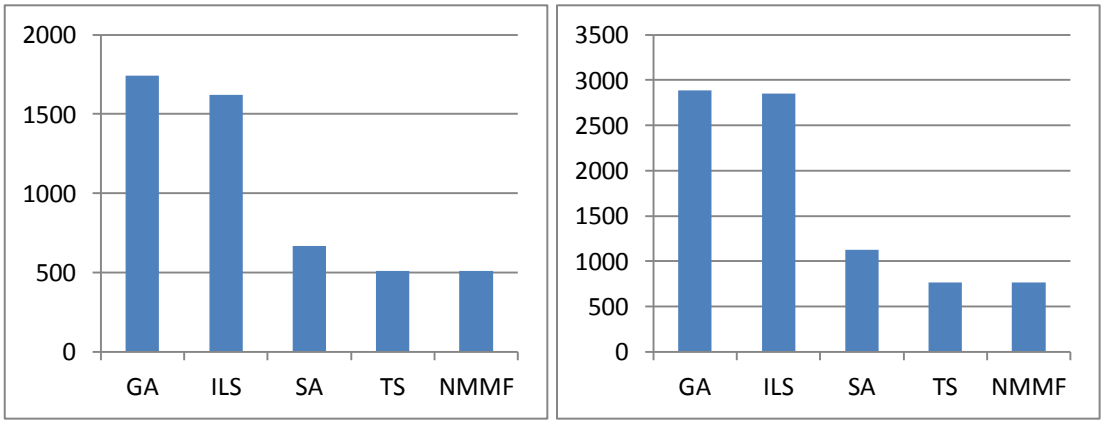


Figure 4.7 Histograms for average performance of *dre30* (left) and *dre42* (right) by the five methods

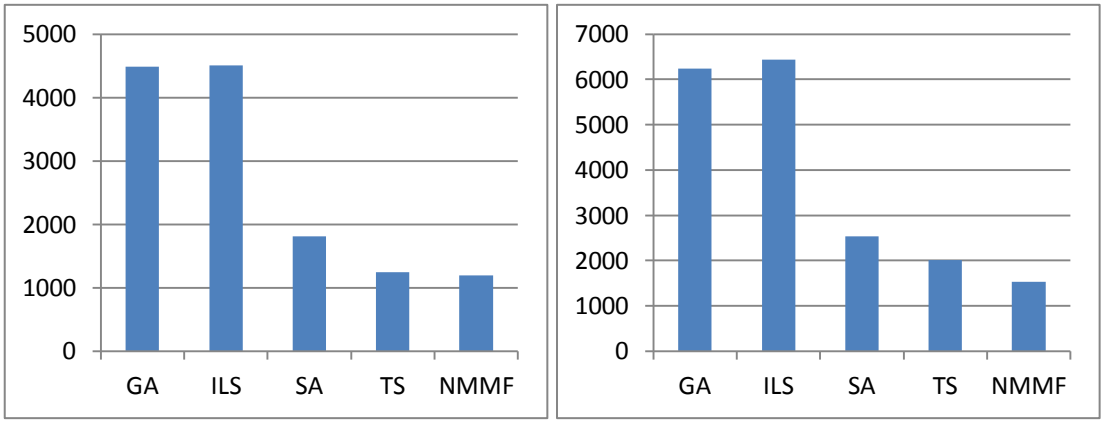


Figure 4.8 Histograms for average performance of *dre56* (left) and *dre72* (right) by the five methods

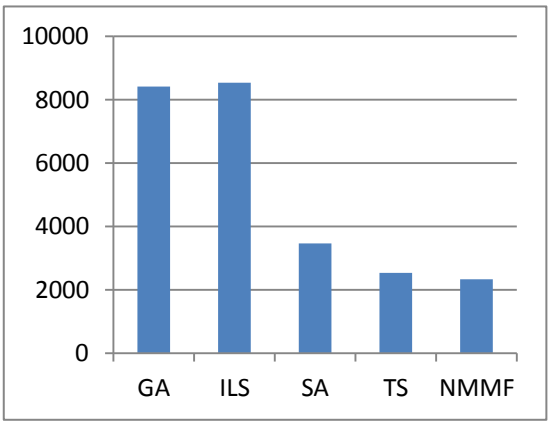


Figure 4.9 Histogram for average performance of *dre90* by the five methods

4.6 Summary

In this chapter, details about the implementation and simulations of NMMF are presented. NMMF is designed for combinatorial optimization problems to achieve flexible search dynamics that enhances the collaborations between different memes. Different

characteristics of memes allow them to achieve healthy balance between exploration and exploitation dynamically. In this chapter the NMMF is presented with details on the training data generation system and the neural network. Conventional optimization approaches usually involve generating or modifying methods or memes manually based on observations and experience. With NMMF, both the collection and incorporation of useful information based on experience collected from earlier problem-solving sessions are carried out automatically. The training data generation system acts as a training dataset supplier for the neural network; it generates optimized parameters sets or mechanisms for groups of relatively small-scale problem instances which is analogous to the experience collected from problem-solving sessions. The neural network is trained based on the features of these small-scale instances and their associated optimized mechanisms. Through the training procedures, the experience collected from solving those small-scale instances is stored in the form of a fully trained neural network. With this neural network, NMMF is capable of generating different optimized meta-meme approaches for both small- and large-scale instances.

Experimental results showed that with selected groups of relatively small-scale QAP instances, the training data generation system produced appropriate parameters for the search control mechanisms at reasonable computational time cost. The features of these instances and their corresponding refined parameters sets were used to train the neural network. The NMMF with fully trained neural network were tested with 135 QAP benchmarks. The experimental results were compared with the results on the same

benchmarks by ES-ILS (Stützle 2006), arguably the best performing method for QAP reported in literature. NMMF outperforms ES-ILS for most of the cases with great improvement on solutions quality and success rates. NMMF even manages to find better solutions for *tai50a* and *tai100a* than the optimum solutions adopted in (Stützle 2006) (not the up to date best known solutions). The results presented by the optimized meta-meme methods generated from NMMF were also compared with the results by some "non-optimized" methods. The comparison demonstrates the superiority of the control parameters generated from the neural network. The capability of generating unique optimized meta-meme methods for different combinatorial optimization problems also improved the consistency and robustness on the performance of NMMF for solving combinatorial optimization problems.

NMMF can be extended by including more memes with necessary modifications, for example, the particle swarming and ant colony optimization. More memes will introduce greater diversity and variety, enhancing the robustness and problem-solving capacity. The greedy search method in the training data generation system tends to incur long computational time for large-scale problems. On this note, heuristic method to assist in the search process will be desirable.

Chapter 5 Solution Fragments as Memes

5.1 Introduction

In this chapter, a different perspective of memes in problem-solving is considered. In the context of solving permutation based problems, a partial solution as fragment that forms part of an overall solution can offer certain advantages when repeated in other scenarios of problem solving. In particular recurring patterns mined during problem solving can serve as useful memes for subsequent instances of solving other problems. The information contained in such a fragment is transmissible between individuals and fits the definition of an evolutionary replicator. Here a meme to its host solution permutation is just the same as for a gene to the chromosome which it is embedded in. As discussed in Chapter 2, inspired by the utilization of the experience from earlier problem-solving sessions in NMMF, the existing solutions of solved problems can be a basis for useful memes, containing the knowledge capable of enhancing the performance in solving those untested problems. Similar to the utilization of the linkages between problems and their favored search methodologies in NMMF, the linkages between optimum solutions and

search spaces of different problems provide theoretical evident for the idea of utilizing an existing solution of one problem as a meme to help solve another unsolved problem. In this chapter, the method of utilizing existing solutions as memes to help solve QAPs is discussed with details on implementation and experimental results.

In the next section, the procedures of comparison and selection of meme providers for an unsolved problem are discussed. In Section 5.3, empirical study of the method with QAP instances is presented. In Section 5.4, the chapter summary is given.

5.2 Extraction of Solution Fragments

At the beginning of the search, random constructed individuals are normally desirable when no prior information is known about the optimum solution of an optimization problem. In other words, since there is no prior knowledge about the optimum solution available and the search space is immense, any initial individual solution constructed based on biased perspective may increase the possibility of bringing that individual further from the optimum solution. The meme or information extraction method presented in this section is responsible for grabbing useful information from existing solutions collected from earlier problem-solving sessions, especially for those problems of equal or smaller dimension, compared to the problem under test. The idea is to compare the data matrices of the two QAPs (one is solved; the other is to be solved). If

the matrices exhibit enough similarity, there is a high chance that the optimum solutions of the two QAPs are similar. The existing optimum solution of the solved QAP can serve as a meme which contains positively biased information in finding the optimum solution of the unsolved one. The solved QAP will be selected as a suitable meme provider for the unsolved QAP. The corresponding optimum s olution will be used as a meme in the form of a solution fragment during construction of the initial population for the unsolved QAP. The method of calculating the similarity level of two matrices is presented in 5.2.1.

5.2.1 Similarity Level

Given a matrix with n elements, and assume that there are n_{equal} elements having the same orders with corresponding elements in another matrix with the same size. The similarity level of the two matrices can be calculated by Eq. (5.1):

$$Similarity\ Level = \frac{n_{equal}}{n}. \tag{5.1}$$

Four rules are defined for calculating similarity level:

1. Similarity level is only applicable for two matrices of the same dimension.

2. If the number of elements with the same value in either the flow matrix or the dominance matrix exceeds a predefined level, the corresponding problem is considered unsuitable as a meme provider for any other QAP.

3. The elements in the two matrices being compared are ranked in ascending order for comparison purpose; the positional index of each element will be its order. The

ranking can also be done in descending way, as long as the two matrices coincide with each other.

4. If two or more elements in a matrix have the same value, all their positional indices after being ranked are allocated to each of them as a group.

The second rule is to avoid the usage of QAPs with large proportion of elements with the same values which makes the corresponding problems trivial in providing memes to help solve other problems. A simple example is provided to explain rules number 3 and 4:

Given matrices *A* and *B*, both with 4 elements:

$A = \{a_{11}=8, a_{12}=11, a_{21}=2, a_{22}=30\}$;

$B = \{b_{11}=12, b_{12}=7, b_{21}=5, b_{22}=7\}$.

To calculate the similarity level, the following steps are performed.

Step 1: Two permutations are ranked in ascending order:

$A_{rank} = \{a_{21}=2, a_{11}=8, a_{12}=11, a_{22}=30\}$;

$B_{rank} = \{b_{21}=5, b_{12}=7, b_{22}=7, b_{11}=12\}$.

Step 2: With the ranked permutations, the order of every element can be easily derived:

$O_A = \{o_{a11}= (2), o_{a12}= (3), o_{a21}= (1), o_{a22}= (4)\}$;

O_B = {o_{b11}= (4), o_{b12}= (2, 3), o_{b21}= (1), o_{b22}= (2, 3)}.

Step 3: Since b_{12} and b_{22} equal to each other, the order for each of them is (2, 3) which contains the positional indices of the two on the ranked permutation. The order of the corresponding elements of the two permutations is compared with each other:

$o_{a11} \neq o_{b11}$; $o_{a12} = o_{b12}$ (The ranking for o_{b12} is (2, 3)); $o_{a21} = o_{b21}$; $o_{a22} \neq o_{b22}$.

Step 4: There are 2 elements in both matrices with the same order among the 4 elements, the similarity level of matrices *A* and *B* is $\frac{2}{4} = 0.5$.

To identify the similarity level of two QAPs, both the flow and distance matrices of the two QAPs are compared based on the above 4 steps. It is evident that the similarity level of each pair of matrices corresponding to the two problems has to be above a preset value in order to adopt one problem as a meme provider for the other,

5.2.2 Partition of Data Matrix

To check whether a candidate QAP is qualified to be a meme provider for an unsolved QAP, the flow and distance matrices of the two need to be compared to calculate their similarity levels. If the dimensions of the two are equal, the four steps in the example in Section 5.2.1 will be carried out for the matrices of the two QAPs. If the size of the candidate QAP is smaller than the unsolved QAP, extra procedures are required to split the matrices of the unsolved QAP of larger dimension into sub-matrices with the same

size as for the candidate QAP. Then the comparison can proceed with the sub-matrices of the unsolved QAP and the matrices of the candidate QAP through the four steps outlined in the last section. In the following, the procedures to split a larger size matrix into several eligible smaller size matrices are discussed. The partition method checks as many elements as possible for an unsolved QAP in order to find more suitable candidate QAPs for providing more memes. The method is described in details with a simple example. Given a QAP of size 6, the task is to divide the flow and distance matrices into sub-matrices of size 3 in order to compare with potential meme providers of size 3. Here only the procedures for splitting the flow matrix are demonstrated; the distance matrix follows exactly the same steps.

Flow matrix

f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
f_{21}	f_{22}	f_{23}	f_{24}	f_{25}	f_{26}
f_{31}	f_{32}	f_{33}	f_{34}	f_{35}	f_{36}
f_{41}	f_{42}	f_{43}	f_{44}	f_{45}	f_{46}
f_{51}	f_{52}	f_{53}	f_{54}	f_{55}	f_{56}
f_{61}	f_{62}	f_{63}	f_{64}	f_{65}	f_{66}

The corresponding solution permutation

s_1	s_2	s_3	s_4	s_5	s_6

Figure 5.1 The flow matrix and the solution permutation of a QAP of size 6.

Step 1: Start with element f_{11} in the upper-left corner on the flow matrix, sub-matrix_1 as shown in Figure 5.2 (the elements selected are highlighted with darkened background) is extracted. In this step, the 9 elements in the upper-left corner are conveniently grabbed to form sub-matrix_1. The corresponding positions on the solution permutation mapped to this sub-matrix are s_1, s_2 and s_3.

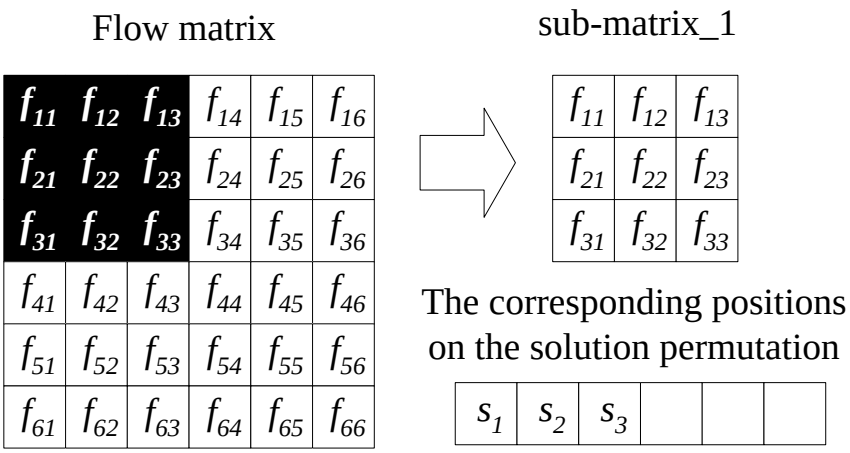


Figure 5.2 The extraction of sub-matrix_1 from the flow matrix.

Step 2: The elements in the first row and column are explored further, since the explored elements are all correlated to f_{11}, this element is again used to form the sub-matrix_2. The positions on the solution permutation mapped from this sub-matrix are s_1, s_4 and s_5.

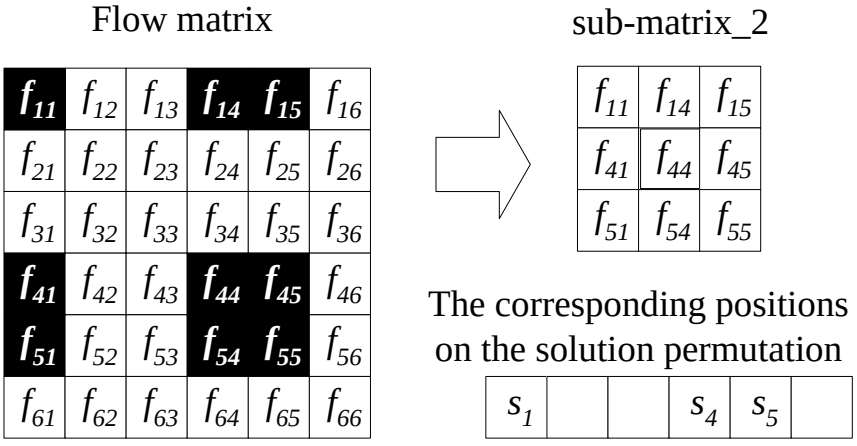


Figure 5.3 The extraction of sub-matrix_2 from the flow matrix.

Step 3: After *Step 2*, there are only two unchecked elements f_{16} and f_{61} left in the first row and column. There are not enough remaining elements to form a sub-matrix of size 3. The elements in the second row and column are explored in this step. This time f_{22} is selected as the upper-left corner element on the sub-matrix. The corresponding positions

for this sub-matrix_3 on the solution permutation are s_2, s_3 and s_4.

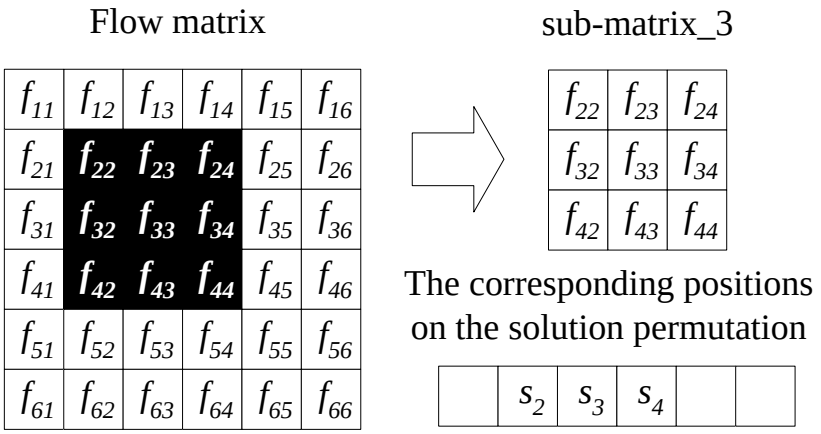


Figure 5.4 The extraction of sub-matrix_3 from the flow matrix.

Step 4: The elements on the second row and column are further explored. The sub-matrix_4 corresponds to positions s_2, s_5 and s_6 on the solution permutation.

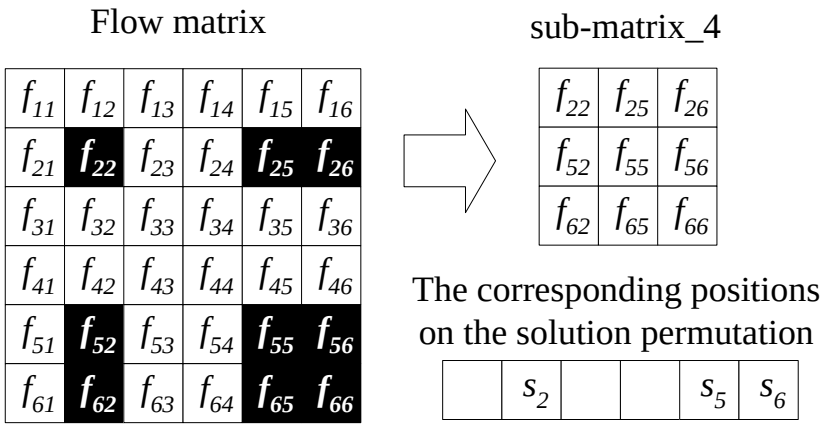


Figure 5.5 The extraction of sub-matrix_4 from the flow matrix.

Step 5: In this step, the third row and column are explored. The elements formed sub-matrix_5 which is mapped to s_3, s_4 and s_5 on the solution permutation.

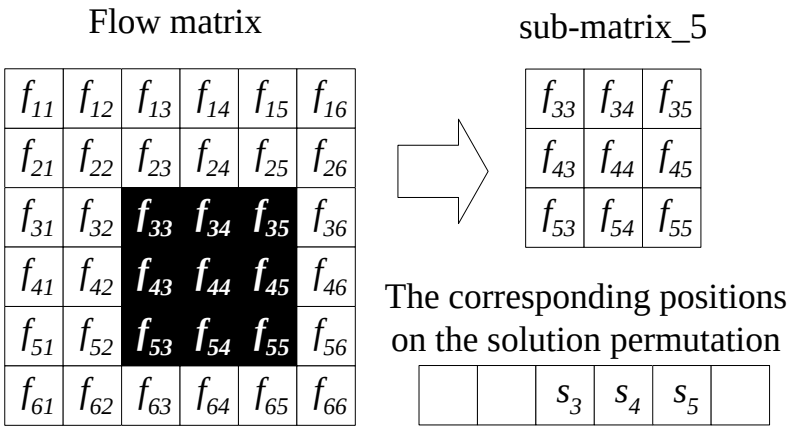


Figure 5.6 The extraction of sub-matrix_5 from the flow matrix.

Step 6: In this step the sub-matrix_6 is extracted. It is mapped to positions s_4, s_5 and s_6 on the solution permutation.

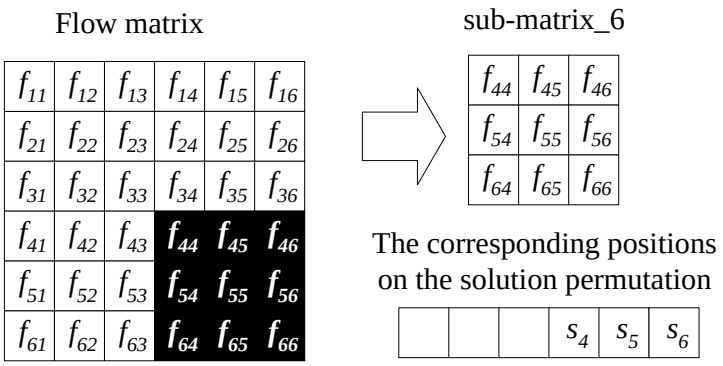


Figure 5.7 The extraction of sub-matrix_6 from the flow matrix.

This partition method scans through every row and column to collect enough number of elements to form sub-matrices which can be eligibly mapped to the elements sets on the solution permutation. With this method, the number of sub-matrices extracted from the main matrix can be calculated as:

$$n_{sub} = \frac{n_1-1}{n_2-1} + \frac{n_1-2}{n_2-1} + \frac{n_1-3}{n_2-1} + \cdots + \frac{n_2-1}{n_2-1}. \quad (5.2)$$

Where n_1 is the size of the main matrix; n_2 is the size of the sub-matrices; $n_1 > n_2$. Each

summand on the right side of Eq. (5.2) is rounded down to an integer. For example, to calculate the number of sub-matrices for the case in the example, n_1 = 6 and n_2 = 3 are substituted into Eq. (5.2):

$$n_{sub} = \frac{6-1}{3-1} + \frac{5-1}{3-1} + \frac{4-1}{3-1} + \frac{3-1}{3-1} = 2.5 + 2 + 1.5 + 1 = 2 + 2 + 1 + 1 = 6. \qquad (5.3)$$

With Eq. (5.2) the number of sub-matrices which can be extracted from a larger size matrix can be easily calculated. The corresponding elements on the solution permutation which the sub-matrices are linked with can be easily located. With this linkage the optimum solution for the smaller size QAP can be converted into a meme in the form of a fragment on the initial solution permutation for the larger size QAP. The details of the conversion are elaborated with the above example:

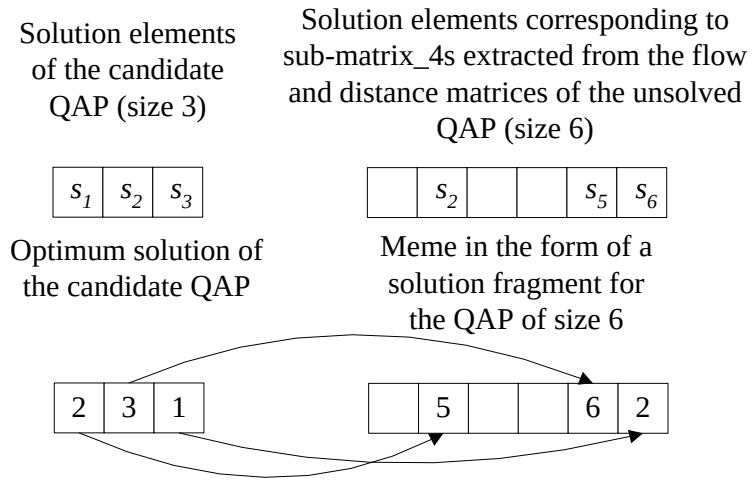


Figure 5.8 Comparison of matrices and conversion of a solution into a meme.

The left side of Figure 5.8 is the solution permutation of the qualified candidate QAP of size 3. The right side of Figure 5.8 is the elements {s_2, s_5, s_6} on solution permutation corresponding to the two sub-matrix_4s extracted from both the flow (refer to Figure 5.5)

and the distance matrices of the unsolved QAP with size of 6. Assume that the order of the elements in the two data matrices of the candidate QAP is compared with that of the sub-matrix_4s of the unsolved QAP and the candidate is qualified to provide meme for the latter. Also assume the existing optimum solution of the candidate QAP is {2, 3, 1}, since the solution permutation {s_1, s_2, s_3} for the candidate QAP corresponds to the elements {s_2, s_5, s_6} mapped to sub-matrix_4s of the unsolved QAP, the solution {2, 3, 1} becomes {5, 6, 2} accordingly. After conversion, the uncompleted initial solution is {x, 5, x, x, 6, 2} for the unsolved QAP, with "x" denoting that the location is not occupied yet. In the last step, all the unallocated facilities and locations are randomly allocated to complete the initialization of this individual solution.

Using multiple memes to construct one individual solution may cause conflict since two different memes may contain the same elements on one solution permutation. For example, if both sub-matrix_5s and sub-matrix_6s (refer to Figures 5.6 and 5.7) are compared with the same candidate and the candidate solution is qualified to provide two memes corresponding to the sub-matrices. If the two memes are embedded on one initial individual solution, the elements on the two are conflicting since the meme associated with sub-matrix_5s is mapped to solution elements {s_3, s_4, s_5} and the meme associated with sub-matrix_6s is mapped to {s_4, s_5, s_6}. Note that locations 4, 5 and facilities 4, 5 cannot allocate twice on one permutation. In such case, the locations 4, 5 and facilities 4, 5 will not be handled in the second meme if the two memes are provided by the same candidate solution permutation.

The maximum number of individuals allowed to be constructed with memes embedded is set to 10 (with respect to a population size of 60). The purpose of introducing such memes into the initial solutions is to guide the search direction in a more efficient way, enhancing the chance of uncovering the global optimum solution. But too many individuals with embedded memes may incur the risk of bringing the search into some local optima and cause premature convergence of the population since there is a high chance that most of such memes lead to overexploitation.

5.3 Experimental Results and Analysis

With the detailed procedures introduced in Section 5.2, every instance from a suite of 135 QAPs is checked to find instance(s) with enough similarity level (with size less or equal to that instance) in the same suite. Incorporation of prior knowledge, the minimum similarity level is set to 0.7, that is, to utilize the solution of one QAP as a meme or solution fragment for another QAP, both of their flow and distance matrices or sub-matrices need to have more than 70% of the elements with the same orders. Also with the rules defined in Section 5.2.1, any QAP with both its flow and distance matrices having more than 70% of the elements with the same values is not qualified to be a meme provider for any other QAPs. With these conditions, qualified meme providers were found for 77 instances.

Table 5.1 Comparison of results for the QAP instances solved with and without usage of embedded memes with a similarity level criterion of 0.7

Benchmark	Optimum Solution	Number of solution(s) with embedded meme(s)	CPU time by NMMF with populations with embedded meme(s)			CPU time by NMMF with randomly initialized population		
			Min time(s)	Ave time(s)	Max time(s)	Min time(s)	Ave time(s)	Max time(s)
bur26a	5426670	1	0.00	**0.03**	0.06	0.03	0.21	1.20
bur26b	3817852	1	0.08	**0.09**	0.11	0.03	0.46	1.92
bur26c	5426795	1	0.06	**0.11**	0.33	0.05	0.19	0.59
bur26d	3821225	1	0.03	**0.22**	1.80	0.01	0.37	3.39
bur26e	5386879	1	0.08	**0.16**	0.86	0.08	0.21	0.91
bur26f	3782044	1	0.05	**0.35**	1.30	0.06	0.42	1.64
bur26g	10117172	1	0.06	**0.17**	0.53	0.08	0.18	0.77
bur26h	7098658	1	0.06	**0.09**	0.14	0.08	0.16	0.31
chr15a	9896	2	0.06	**0.12**	0.33	0.06	0.14	0.55
chr20a	2192	1	0.11	1.17	3.39	0.06	**0.93**	4.30
chr20b	2298	1	0.23	**7.43**	27.47	0.17	8.00	31.38
chr22a	6156	1	0.19	**0.83**	2.55	0.09	0.98	2.08
chr22b	6194	1	0.19	**1.56**	4.61	0.52	1.81	4.25
els19	17212548	10	0.03	**0.17**	0.38	0.03	0.26	0.53
esc128	64	10	0.64	**1.45**	1.67	0.66	1.48	1.72
esc32a	130	10	0.13	**2.48**	5.44	0.16	3.06	13.94
kra30a	88900	9	0.16	**0.54**	1.91	0.25	0.70	1.76
kra30b	91420	9	0.20	**0.65**	2.16	0.20	0.80	2.20
kra32	88700	10	0.09	**0.49**	1.75	0.20	0.70	1.72
lipa30a	13178	1	0.13	**0.24**	0.31	0.13	0.26	0.48
lipa30b	151426	1	0.09	**0.12**	0.19	0.11	0.15	0.23
lipa40a	31538	1	0.50	1.07	2.25	0.45	**1.01**	2.16
lipa40b	476581	1	0.17	**0.29**	0.52	0.16	**0.29**	0.41
lipa50a	62093	1	1.53	**3.74**	5.89	2.19	4.01	6.53
lipa50b	1210244	1	0.38	0.53	0.88	0.39	**0.50**	0.64
lipa60a	107218	1	4.05	**10.55**	18.67	5.03	12.07	20.70
lipa60b	2520135	1	1.02	**1.43**	2.13	0.75	**1.43**	2.95
lipa70a	169755	1	10.13	**27.16**	38.69	14.22	31.94	101.44
lipa70b	4603200	1	1.17	**2.11**	2.95	1.09	2.39	3.70
lipa80a	253195	1	22.17	**112.28**	252.22	82.42	141.39	222.41
lipa80b	7763962	1	2.44	**4.34**	6.91	1.92	4.67	8.19
lipa90a	360630	1	79.23	**237.51**	415.80	56.22	285.71	587.11
lipa90b	12490441	1	3.45	**6.75**	14.63	3.58	9.02	15.94
nug30	6124	10	0.16	0.42	1.05	0.14	**0.39**	1.28
rou20	725522	2	0.14	**0.41**	0.98	0.06	0.42	1.73
scr20	110030	9	0.05	**0.10**	0.14	0.08	0.13	0.20
sko72	66256	3	45.36	111.36	212.17	37.00	**97.19**	201.00
sko81	90998	3	25.30	**207.45**	572.70	40.02	276.61	978.51
ste36a	9526	7	0.27	**0.95**	2.78	0.70	3.91	11.98
ste36b	15852	7	0.05	**0.09**	0.14	0.27	1.38	2.94
ste36c	8239110	7	0.09	**0.10**	0.11	1.31	2.64	4.13

These 77 instances were tested by NMMF presented in Chapter 4 with initial solution(s) constructed with embedded memes provided by optimum solutions of the qualified

meme providers. In the current implementation, the meme(s) embedded in one initial individual solution are provided by only one solution of a candidate, this is to make the experiment simple and straightforward. Each instance was tested for 20 runs. These instances were also tested by NMMF with randomly initialized populations (without usage of the embedded memes) with the same settings. The optimum solutions were uncovered in all 20 runs in both situations for all 77 instances. The parametric settings of the NMMF and computing platform are the same as in Chapter 4.

The CPU time costs for the 41 benchmarks under the two situations by NMMF are listed in Table 5.1. Those instances which can be solved within 0.1 seconds on average are not included. *Number of solutions with embedded memes* column refers to the number of solutions with embedded memes in the form of solution fragments in the initial population. Other terms were explained in Chapters 3 and 4.

The better average time cost for each benchmark is highlighted in bold. The comparison in Table 5.1 shows that out of the 41 instances, the average CPU time costs are reduced with the usage of embedded memes in 34 of them. For *ste36a*, *ste36b* and *ste36c*, the average CPU time costs are reduced from 3.91, 1.38 and 2.64 seconds to 0.95, 0.09 and 0.10 seconds respectively, computational time costs for the latter two cases are improved for more than 10 times. The results show that the adoption of embedded memes in the initial population brings positive effect for most of the cases.

Next the minimum similarity level is decreased to 0.5, qualified candidate meme providers were found for 23 more cases. With the decrement of the criterion value, the

selected memes have the tendency to provide more random information than those selected with a higher similarity level. Experiments were carried out for these cases just as for the 41 cases shown in Table 5.1. The results are shown in Table 5.2 for both with and without the adoption of the selected memes.

Table 5.2 Comparison of results for the QAP instances solved with and without usage of memes selected with a similarity level criterion of 0.5

Benchmark	Optimum Solution	Number of solution(s) with embedded meme(s)	CPU time by NMMF with initial populations embedded with meme(s)			CPU time by NMMF with randomly initialized population		
			Success rate(%)	Ave gap(%)	Ave time(s)	Success rate(%)	Ave gap(%)	Ave time(s)
sko42	15812	10	100	0	**2.32**	100	0	2.43
sko49	23386	10	100	0	28.68	100	0	**18.99**
sko56	34458	10	100	0	19.44	100	0	**17.93**
sko64	48498	10	100	0	**27.23**	100	0	32.26
sko90	115534	10	85	0.0057	**273.01**	**90**	**0.0038**	281.44
Sko100a	152002	10	70	**0.0080**	403.37	**75**	0.0088	**309.73**
sko100b	153890	10	100	0	331.05	100	0	**276.87**
sko100c	147862	10	90	0.00014	**401.41**	**95**	**0.000068**	406.94
sko100d	149576	10	**90**	**0.0025**	**296.94**	75	0.0072	391.51
sko100e	149150	10	85	0.00060	553.29	**90**	**0.00047**	**492.28**
sko100f	149036	10	75	0.0048	**362.30**	**80**	0.0048	420.21
tai40b	637250948	10	100	0	2.36	100	0	**1.94**
tai50b	458821517	10	100	0	**14.84**	100	0	17.25
tai60b	608215054	10	100	0	**28.14**	100	0	28.80
tai80b	818415043	10	100	0	165.43	100	0	**150.06**
tai100b	1185996137	10	**85**	**0.011**	286.06	80	0.025	**261.15**
tai150b	498896643	10	0	0.39	982.11	0	0.33	**976.53**
tai64c	1855928	8	100	0	**0.16**	100	0	0.20
tho30	149936	10	100	0	0.49	100	0	**0.41**
tho40	240516	10	100	0	19.40	100	0	**18.34**
tho150	8133398	10	0	0.040	**822.89**	**5**	**0.039**	893.01
wil50	48816	9	100	0	**34.72**	100	0	42.37
wil100	273038	9	**30**	0.0017	298.39	25	0.0036	**259.99**

The results for these 23 cases are presented in Table 5.2 for comparison. Similarly, the better success rate, average gap and average time cost for each case are highlighted in bold. In a summary, the average computational time costs are reduced slightly only in 11 out of 23 cases with the adoption of memes in the form of the solution fragments; for the other 12 cases the average time costs are even higher. Since the success rates and average

solution costs are quite close to each other for all the 23 cases, it can be considered that there is no improvement or deterioration with the usage of the solution fragments. The results indicate that the memes selected with a minimum similarity level which is less than a certain value do not necessarily decrease the computational time cost. In other words, such memes do not improve the search efficiency. The simulation results on these 23 cases met the expectation that the solution fragments provided by problems which do not have enough similarity level with the unsolved problem are providing more random information rather than useful "shortcuts".

Based on the observation of the simulation results in Tables 5.1 and 5.2, the solutions of the solved problems are indicative of improvement in search efficiency for most of the cases with a certain selection criterion. This met the expectation that the existing solutions of the solved problems could bring useful information to the initial search space of a new problem. As discussed in Section 5.1 and 5.2, the data and solution structure of the QAP problem set provide the possibility of reutilizing existing solution strings. QAP is a typical assignment problem and the cost value is calculated directly based on the solution permutation plus the flow and distance matrices. The order of the elements in the distance and flow matrices directly affects the optimum solution permutation with the minimum cost. If the orders are similar for the matrices of two QAPs, their optimum solution permutations should have some level of similarity, in other words, their solution strings must have notable number of elements with the same values. This relationship suggests that the optimum solution of one problem is very possibly a good solution for

another problem if the two have a high similarity level, in other words, the optimum solution of one is very possibly close to the optimum solution of another in terms of the hamming distance. When such a solution is adopted in the initial search space of the unsolved problem, it could provide a much shorter path to the optimum solution compared to the random staring points. Therefore, the search efficiency is improved. In summary, this method successfully utilizes the existing solutions and provides better results compared to the search without any prior information included in the initial search space.

5.4 Summary

In this chapter, the method of utilizing the existing solutions as memes embedded in the initial populations for the unsolved QAP instances is presented and tested. Compared to a randomly initialized population, the population with embedded meme(s) selected based on some selection criteria has the potential of improving the search efficiency of a search method, finding good quality or even the optimum solutions for large-scale QAPs in relatively shorter computational time cost. The empirical study on 41 QAP benchmarks shows that computational time costs are reduced for most of the cases with the usage of the selected memes. Extensive testing on 23 cases shows that the memes selected with a decreased similarity level criterion do not help to increase the search efficiency. This fact indicates that such a meme will only bring positive effect for a search if the

corresponding meme provider compared to the unsolved problem instance has a similarity level above a certain level.

One possible negative effect of this method is the high chance of premature convergence of the population because all memes provided from the qualified QAPs contain only local information with respect to the unsolved QAP. This will further lead the search result to some local optima instead of the global optimum solution. In future this potential problem could be addressed if a method to arrange the memes on the solution permutations with strategies incorporated to balance the local and global information.

Chapter 6 Conclusions and Future Research Directions

6.1 Conclusions

As discussed in Chapter 2, for a hybrid or meta-meme method, balancing between diversification and intensification, or exploration and exploitation, is the decisive factor in achieving good and reliable search performance. Meta-meme approaches usually have been shown to bring about greater flexibility in managing the healthy balance between exploration and exploitation. In this thesis, different techniques of managing and incorporating memes are presented with details of implementations and validation through experimental results.

In the proposed collaborative memes framework, different memes were allocated with their private search spaces. Different memes have different search routines resulting in different levels of balance between exploration and exploitation. The private search spaces are designed to preserve their unique search specialties. The communication protocol between memes brings about certain level of collaboration which guarantees the

robustness of the approach. The option for drastic perturbation is introduced to enhance the level of exploration when necessary. All these strategies function to strike a balance between high population diversity (exploration) and better solution quality (exploitation). The CMF and the independent memes of the CMF were tested using several groups of large-scale QAP instances. The comparison between the experimental results by CMF and the three memes executing independently shows the robustness of CMF with convincingly better performance. The results further prove that the scheme is capable of incorporating multiple memes to integrate their merits in a desirable manner, resulting in a more robust search methodology. The results of CMF are also compared with two other robust methods, the hybridized genetic algorithm and the parallel memetic algorithm with diversity-based dynamic adaptive strategy. The comparison confirms the competitiveness of CMF in solving QAPs.

Different from CMF which manages memes in a rather static manner, the neural meta-memes framework is a problem-adaptive search methodology which manages memes dynamically. This framework is designed with the capability of producing different optimization recipes for managing memes to construct optimized search methodologies for different problems. This problem-adaptive merit is achieved through the neural network which captures the knowledge about the linkages between memes and problems. Such knowledge is acquired by training the neural network with the experience collected from earlier problem-solving sessions.

The neural meta-memes framework was tested with a suite of QAP instances. The neural

network was trained with training data generated with groups of relatively small-scale problems. The framework with the trained neural network generated optimized search methodologies which were applied to solve untested problems, including the large-scale QAP instances. The results were compared with ES-ILS which is probably one of the most competitive methods for QAP and has been tested and shown to produce good results for most of the QAP instances. The comparison shows the robustness of the framework. In the extensive testing on a group of benchmarks which are claimed to be difficult for stochastic methods, the framework outperforms all its constituent memes running independently.

Neural meta-memes framework takes advantage of the linkages between different problems and their associated favored search methodologies; hence the experience collected from earlier problem-solving sessions becomes useful information to help construct a robust mete-memes method for a specific problem. Similarly, earlier problem-solving sessions are utilized in a different manner, this time the linkages between search spaces of different problems and their associated solution permutations were studied. If two problems are structurally similar and one of them is already solved, there is a very high possibility that the solution of the solved one contains useful information that can be used in solving the untested problem. Based on this theoretical premise, a method was implemented to adopt the existing solution permutations as memes embedded in the initial solutions to tackle an unsolved problem. The similarity level between search spaces of two problems is the only criterion for the selection of

qualified candidate meme provider(s) for a problem. This method was tested with NMMF on groups of QAP benchmarks. Simulation results produced with the usage of such memes were compared with the results produced with randomly constructed initial solutions. The comparison shows that the computational time costs for most of the cases were reduced with the usage of the memes. Extensive testing on groups of benchmarks with meme providers selected with a lower similarity level did not show obvious reduction in the computational time cost. This met the expectation that if the meme providers do not have significant similarity level with the unsolved problem, the selected memes do not serve to enhance the performance of a search method.

6.2 Future Research Directions

Although the two approaches, CMF and NMMF, show their robustness through simulations with large-scale combinatorial optimization problems, it is evident that both of the methods can still be enriched by including more diverse memes or search methods, even exact search methods. Such enrichment will equip the two search methodologies with much wider range of capability for handling different types of optimization problems.

The empirical studies of the two approaches have been confined to QAP instances. Applying on other types of combinatorial optimization problems, such as the job-shop

scheduling problem, vehicle routing problem, etc., should be studied.

Furthermore, the population diversity with the NMMF is not adopted as a decisive parameter to help in maintaining balance between exploitation and exploration. With proper implementation, the diversity value could be used as a criterion to justify whether some necessary corresponding actions should be carried out to maintain a certain level of diversity to prevent a premature convergence situation. With this implementation, NMMF is likely to be more robust.

As for the method of utilizing solutions as memes for solving QAPs, currently one initial solution only accepts memes from one solution of a candidate. This means that the fragment elements for one initial solution are all provided by one solution permutation string. One viable research direction is to study how one solution can be initialized with multiple memes from different solutions of different individual candidates. This could bring greater variety to the initial population. The possible negative effects such as the possible premature convergence brought by the memes to the search have not been studied. Other types of permutation-based combinatorial optimization problems could be helpful on this point. If we have a better understanding on the possible negative effects of this method, maintaining a proportionate and desirable balance between the individuals with embedded memes and the random initialized individuals could be a good research area. Future study on such issues could be beneficial. Additionally, if the memes can be distributed in the initial population with strategies related to the features of their corresponding candidate problems, the potential for further enhancement in performance

is likely. Currently, the work reported applies only to permutation-based combinatorial optimization problems. In future, the method can be applied to other types of problems such as problems based on binary representation as long as the linkages between their search spaces and solutions exist. Furthermore, the prospect of the approach being applicable to continuous domain problems is an interesting area of future research focus.

Chapter 7 Author's Publications

7.1 Journals

1. Song L. Q., Lim M. H. and Suganthan P. N. (2010) "Ensemble of Optimization Algorithms for Solving Quadratic Assignment Problems," *International Journal of Innovative Computing, Information and Control*, vol. 6, no. 9.

2. Song L. Q., Lim M. H. and Ong Y. S. "Neural Meta-Memes Framework for Managing Search Algorithms in Combinatorial Optimization," *European Journal of Operational Research*, (submitted).

7.2 Book Chapter

1. Song L. Q., Lim M. H. and Ong Y. S., "Neural Meta-Memes Framework for Combinatorial Optimization," *Swarm, Evolutionary, and Memetic Computing Lecture Notes in Computer Science*, vol. 6466/2010, pp. 198-205, 2010.

7.3 Conferences

1. Chen X., Lim M. H., Ong Y.S. and Song L., "An Ant Colony System Algorithm for Path Planning in Sparse Graphs," *Int'l Conf on Intelligent and Advanced Systems*, Nov. 2007, KL, Malaysia.

2. Tang J., Song L., Lim M. H. and Ong Y. S., "Hierarchical Model Parallel Memetic Algorithm in Heterogeneous Computing Environment", *2007 IEEE Congress on Evolutionary Computation*, CEC 2007, Article number 4424820, Pages 2758-2765, Sept. 2007.

3. Song L. Q. and Lim M. H., "Ensemble for Solving Quadratic Assignment Problems," *International Conference on Soft Computing and Pattern recognition*, December 2009, Malacca, Malaysia.

4. Cao Q., Song L. Q., Shi X. and Li J. H., "Multi-Tasking of Fuzzy Inference Processor Through Real-Time Context Switching," *2009 IEEE/ASM E International Conference on Advanced Intelligent Mechatronics*, July 2009, Singapore.

5. Song L. Q., "Utilization of Chromosome Fragments on Solving Quadratic Assignment Problems," *International Conference on Computational Problem-Solving*, Li Jiang, China, Dec. 2010.

6. Song L. Q., Lim M. H. and Ong Y. S. "Neural Meta-Memes Framework for Managing Search Algorithms in Combinatorial Optimization," *IEEE Symposium Series on Computational Intelligence*, Paris, France, Apr. 2011.

Bibliography

Abramson D., Krishnamoorthy M. and Dang, H. (1999) “Simulated annealing cooling schedules for the school timetabling problem,” *Asia-Pacific Journal of Operational Research*, vol. 16, pp. 1-22.

Aunger R. (Ed.) (2000) *Darwinizing culture – the status of memetics as a science*, Oxford University Press.

Azim G. A. (2006) “Neural Networks for solving the quadratic assignment problem,” *Neural Information processing - letters and reviews*, vol. 10. no. 3, pp. 51-56.

Bachelet V., Preux P. and Talbi E.-G. (1996) “Parallel hybrid metaheuristics: application to the quadratic assignment problem," In: *Proceedings of the Parallel Optimization Colloquium*, Versailles, France.

Back T., Fogel D. B., and Michalewicz Z. (Eds.) (1997) *Handbook of Evolutionary Computation*, Institute of Physics Publishing Ltd, Bristol, Philadelphia, New York, Oxford University Press, U K.

Baker J. E. (1987) “Reducing bias and inefficiency in the selection algorithm,” In: *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, Cambridge, Massachusetts, United States, pp. 14-21.

Battiti R. and Tecchiolli G. (1994) "The reactive tabu search," *ORSA Journal on Computing*, vol. 6, pp. 12-140.

Blackmore S. (1999) *The meme machine*, New York: Oxford University Press.

Burkard R. E., Karisch S. E. and Rendl F. (1997) "QAPLIB A quadratic assignment problem library," *Journal of Global Optimization*, vol. 10, pp. 391-403. Available from <http://www.opt.math.tu-graz.ac.at/qaplib/>http://qaplib.uwaterloo.ca/inst.html

Carreto C. and Baker B. (2002) "A GRASP interactive approach to the vehicle routing problem with backhauls," In: *Essays and Surveys in Metaheuristics*, Ribeiro C. C. and Hansen P. (Eds.), Kluwer Academic Publishers, pp. 185-199.

Chakrapani J. and Skorin-Kapov J. (1992) "A connectionist approach to the quadratic assignment problem," *Computers and Operations Research*, vol. 19, no. 3-4, pp. 287-295.

Chen S. and Luk B. L. (1999) "Adaptive simulated annealing for optimization in signal processing applications," *Signal Processing*, vol. 79, no. 1, pp. 117-128.

Connolly D. T. (1990) "An improved annealing scheme for the QAP," *European Journal of Operational Research*, vol. 46, pp. 93-100.

Cook S. A. (1971) "The complexity of theorem proving procedures," In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ACM, New York, pp. 151-158.

Cordon O., Fernandez I., Herrera F. and Moreno L. (2000) "A new ACO model integrating evolutionary computation concepts: The best-worst ant system," In: *Abstract Proceedings of ANTS2000 - From Ant Colonies to Artificial Ants: A Series*

of International Workshops on Ant Algorithms, Dorigo M., Middendorf M. and Stützle T. (Eds.), Université Libre de Bruxelles, Belgium, pp. 22–29.

Croes G. A. (1958) "A method for solving traveling salesman ponblems," *Operational Research*, vol. 5, pp. 791-812.

Cung V. D., Mautor T., Michelon P. and Tavares A. (1997) "A scatter search based approach for the quadratic assignment problem," In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC)*, Baeck T., Michalewicz Z. and Yao X. (Eds.), Indianapolis, USA, IEEE Press, pp. 165-170.

Cybenko G. (1989) "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314.

Darwin C. (1859) *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, London: John Murray.

Davis L. (Ed.) (1991) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

Dawkins R. (1976) *The Selfish Gene*, Oxford: Oxford University Press.

De Jong K. A. (1975) "An analysis of the behavior of a class of genetic adaptive systems," *Doctoral dissertation*, University of Michgan, Ann Arbor, Michian.

Delius J. (1989) "Of mind memes and brain bugs, a natural history of culture," In: *The Nature of Culture*, Koch W. A. (Ed.), Bochum, Germany: Bochum, pp. 26–79.

Dennet D. (1991) Consciousness Explained, Boston, MA, Little Brown.

Dickey J. W. and Hopkins J. W. (1972) "Campus building arrangement using TOPAZ," *Transportation Science*, vol. 6, pp. 59-68.

Distin K. (2005) *The Selfish Meme: A Critical Reassessment*, Cambridge University Press.

Dorigo M. (1992) "Optimization, learning and natural algorithms," *PhD thesis*, Department of Systems and Information Electronic Engineering, Politecnico di Milano, Italy, 1992.

Dorigo M., Maniezzo V. and Colorni A. (1996) "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, no. 1, pp. 29-41.

Drezner Z. (2003) "A new genetic algorithm for the quadratic assignment problem," *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 320-330.

Drezner Z. (2005a) "Compounded genetic algorithms for the quadratic assignment problem," *Operations Research Letters*, vol. 33, pp. 475-480.

Drezner Z. (2005b) "The extended concentric tabu for the quadratic assignment problem," *European Journal of Operational Research*, vol. 160, pp. 416-422.

Drezner Z. Hahn P. Taillard E. (2005) "A Study of Quadratic Assignment Problem Instances that are Difficult for Meta-heuristic Methods," *The Annals of Operations Research, Integer Programming: State of the Art and Recent Advances*, vol. 139, pp 65-94.

Duman E. and Or I. (2007) "The quadratic assignment problem in the context of the printed circuit board assembly process," *Computers and operations research*, vol. 34, no. 1, pp. 163-179.

Elshafei A. N. (1977) "Hospital layout as quadratic assignment problem," *Operational*

Research Quarterly, vol. 28, pp. 167-179.

Emden-Weinert T. and Proksch M. (1999) "Best practice simulated annealing for the airline crew scheduling problem," *Journal of Heuristics*, vol. 5, pp. 419-436.

Feo T. A. and Resende M. G. C. (1989) "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol. 8, pp. 67-71.

Fleurent C. and Ferland J. A. (1994) "Genetic hybrids for the quadratic assignment problem," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 16, pp. 173-187.

Flood M. M. (1956) "The traveling-salesman problem," *Operations Research*, vol. 4, pp.61-75.

Fogel L. J. (1962) "Toward inductive inference automata," In: *Proceedings of the International Federation for Information processing Congress*, Munich, pp. 395-399.

Fogel L. J., Owens A. J. and Walsh M. J. (1966) *Artificial Intelligence through Simulated Evolution*, Wiley.

Freisleben B. and Merz P. (1996) "A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," In: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, Back T., Kitano H., and Michalewicz Z. (Eds.), IEEE Press, pp. 616-621.

Gambardella L. M., Taillard E. D. and Dorigo M. (1999a) "Ant colonies for the quadratic assignment problem," *Journal of the Operational Research Society*, vol. 50, pp. 167-176.

Gambardella L.M., Taillard E D. and Agazzi G. (1999b) "MACS-VRPTW: A multiple

ant colony system for vehicle routing problems with time windows," In: *New Ideas in Optimization*, Corne D., Dorigo M. and Glover F. (Eds.), McGraw-Hill, London, pp. 63-76.

Gendreau M., Pesant G. and Rousseau L. M. (2002) "Using constraint based operators with variable neighborhood search to solve the vehicle routing problem with time windows," *Journal of heuristics*, vol. 8, no. 1, pp. 43-59.

Gilmore P. C. (1962) "Optimal and suboptimal algorithms for the quadratic assignment problem," *SIAM Journal on Applied Mathematics*, vol. 10, pp. 305-313.

Glover F. W. (1986) "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549.

Glover F. W. (1989) "Tabu Search – Part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp.190-206.

Glover F. W. (1990) "Tabu Search – Part II," *ORSA Journal on Computin*, vol. 2, no. 1, pp.4-32.

Glover F. W. and Kochenberger G. A. (Eds.). (2003) *Handbook of Metaheuristics*, vol. 57 of International series in operations research and management science. Kluwer Academic Publishers, Boston Hardbound.

Glover F. W. and Laguna M. (1997) *Tabu Search*, Kluwer Academic Publishers.

Goldberg D. E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

Gong T. and Tuson A.L. (2008) "Particle swarm optimization for quadratic assignment problems—a forma analysis approach," *International Journal of Computational*

Intelligence Research, vol. 4, no. 2, pp. 177-185.

Gould S. J. (1991) *Bully for Brontosaurus*, New York, Norton.

Hancock P. J. B. (1994) "An empirical comparison of selection methods in evolutionary algorithms," In: *Evolutionary Computing: AISB Workshop*, Fogarty T. C. (Ed.), Berlin: Springer-Verlag.

Hansen P. and Mladenovic N. (2001) "Variable neighborhood search: Principles and applications," *European Journal of Operational Research*, vol. 130, pp. 449-467.

Hart J. P. and Shogan A. W. (1987) "Semi-greedy heuristics: An empirical study," *Operations Research Letters*, vol. 6, pp. 107-114.

Haykin S. (1999) *Neural Networks: A Comprehensive Foundation* (2nd Ed.). Prentice Hall.

Holland J. H. (1962) "Outline for a logical theory of adaptive systems," *Journal of the ACM*, vol 9, no. 3, pp 297–314.

Holland J. H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Holstein D. and Moscato P. (1999) "Memetic algorithms using guided local search: A case study," In: *New Ideas in Optimization*, Corne D., Dorigo M. and Glover F. (Eds.), McGraw Hill, Berkshire, England, pp. 235–243.

Hopfield J. J. (1982) "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the USA*, vol. 79, no. 8, pp. 2554-2558.

Hull D. L. (1982) "The naked meme," In: Plotkin H. C. (Ed.) *Learning, Development,*

and Culture, Essays in Evolutionary Epistemology, Chicester: John Wiley & Sons Ltd.

Ishii S. and Sato M. (1998) "Constrained neural approaches to quadratic assignment problems," *Neural Networks*, vol. 11, pp. 1073-1082.

Johnson D. S. (1990) "Local optimization and the travelling salesman problem," In: *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, LNCS, Springer, Berlin, vol. 443, pp. 446-461.

Kendall G., Soubegia E. and Cowling P. (2002) "Choice function and random hyperheuristics," In: *Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 667-671.

Kennedy J. and Eberhart R. (1995) "Particle swarm optimization," *Proceedings of the IEEE International Conference on Neural Networks*, Piscataway, NJ, pp. 1942-1948.

Kilby P., Prosser P. and Shaw P. (1999) "Guided local search for the vehicle routing problem with time windows," In: *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Vob S., Martello S., Osman I. and Roucairol C. (Eds.), Kluwer Academic, pp. 473-486.

Kilby P., Prosser P. and Shaw P. (2000) "A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints," *Constraints*, vol. 5, no. 4, pp. 389-414.

Kirkpatrick S., Gelatt C. D. and Vecchi M. P. (1983) "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680.

Kohonen T. (1988) *Self-Organization and Associative Memory*, Springer, Berlin.

Koopmans T. C. and Beckmann M. J. (1957) “Assignment problems and the location of economic activities,” *Econometrica*, vol. 25, pp. 53-76.

Koza J. R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA.

Krarup J. and Pruzan P. M. (1978) “Computer-aided layout design,” *Mathematical Programming Study*, vol. 9, pp. 85-94.

Krasnogor N., Blackburne B., E. K. and Hirst J. D. (2002) “Multimeme algorithms for protein structure prediction. In: *Proceedings of the Parallel Problem Solving from Nature VII, Lecture Notes in Computer Science*, vol. 2439, pp. 769-778.

Krasnogor N. and Gustafson S. (2004) “A study on the use of "self-generation" in memetic algorithms,” *Natural Computing*, vol. 3, pp. 53-76.

Laursen P. S. (1993) “Simulated annealing for the QAP—Optimal tradeoff between simulation time and solution quality,” *European Journal of Operational Research*, vol. 69, pp. 238-243.

Lawler E. L. (1963) “The quadratic assignment problem,” *Management Science*, vol. 9, no. 4, pp. 586-599.

Lawler E. L. (1976) *Combinatorial optimization: Networks and Matroids*, Holt, Rineh art and Winston, New York.

Li Y. and P. M. Pardalos. (1992) “Generating quadratic assignment test problems with known optimal permutations,” *Computational Optimization and Applications*, vol. 1, pp. 163-184.

Li Y., Pardalos P. M. and Resende M. G. C. (1994) “A greedy randomized adaptive

search procedure for the quadratic assignment problem," In: *Quadratic Assignment and Related Problems*, Pardalos P. and Wolkowicz H. (Eds.), AMS: Providence, Rhode Island, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 16, pp. 173-187.

Lim M. H., Gustafson S., Krasnogor N. and Ong Y. S. (2009) "Editorial to the first issue," *Memetic Computing*, vol. 1, pp. 1-2.

Lim M. H., Yuan Y. and Omatu S. (2000) "Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem," *Computational Optimization and Applications*, vol. 15, pp. 249-268.

Lim M. H., Yuan Y. and Omatu S. (2002) "Extensive testing of a hybrid genetic algorithm for quadratic assignment problem," *Computational Optimization and Applications*, vol. 23, pp. 47-64.

Lynch A. (1991) "Thought contagion as abstract evolution," *Journal of Ideas*, vol. 2, pp. 3-10.

Locatelli M. (2000) "Simulated annealing algorithms for continuous global optimization: convergence conditions," *Journal of Optimization Theory and Applications*, vol. 104, pp. 121-133.

Loiola E. M., Abreu N. M. M. de, Boaventura-Netto P. O., Hahn P. and Querido T. (2007) "A survey for the quadratic assignment problem," *European Journal of Operational Research*, vol. 176, no. 2, pp. 657-690.

Lourenco H. R. (1995) "Job-shop scheduling: Computational study of local search and large-step optimization methods," *European Journal of Operational Research*, vol.

83, no. 2, pp. 347-364.

Lourenco H. G., Martin O. C. and Stützle T. (2003) “Iterated local search,” In: *Handbook of Metaheuristics*, Glover F. and Kochenberger G. A. (Eds.), pp. 321-353. Kluwer Academic Publishers.

Maniezzo V. (1999) “Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem,” *INFORMS Journal on Computing*, vol. 11, no. 4, pp. 358-369.

Maniezzo V. and Colorni A. (1999) “The ant system applied to the quadratic assignment problem,” *Knowledge and Data Engineering*, vol. 11, pp. 769-778.

Maniezzo V. and Colorni A. and Dorigo M. (1994) The Ant System Applied to the Quadratic Assignment Problem, Tech. Rep. 94/28, IRIDIA, University de Bruxelles.

Martin O. and Otto S. W. (1995) “Partitioning of unstructured meshes for load balancing,” *Concurrency: Practice and Experience*, vol. 7, pp. 303-314.

Martin O. and Otto S. W. (1996) “Combining simulated annealing and local search heuristics,” *Annals of Operations Research*, vol. 63, pp. 57-75.

Mavridou T. D. and Pardalos P. M. (1997) “Simulated annealing and genetic algorithms for the facility layout problem: A survey,” *Computational Optimization and Applications*, vol. 7, pp. 111-126.

McCulloch W. S. and Pitts W. (1943) “A logical calculus of ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133.

Merkle D., Middendorf M. and Schmeck H. (2000) “Ant colony optimization for resource-constrained project scheduling,” In: *Proceedings of the Genetic and*

Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann Publishers, San Francisco, CA, pp. 893–900.

Merz P. and Freisleben B. (1999) "A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem," In: *Proceedings of the 1999 International Congress of Evolutionary Computation*, IEEE Press, pp. 2063-2070.

Merz P. and Freisleben B. (2000) "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 337-352.

Metropolis N., Rosenbluth A. W., Teller A. H. and Teller E. (1953) "Equation of state calculation by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087-1091.

Meuth R. Lim M. H. Ong Y. S. and Wunsch D. C. (2009) "A proposition on memes and meta-memes in computing for higher-order learning," *Memetic Computing*, vol. 1, no. 2, pp. 85-100.

Middendorf M., Reischle F. and Schmeck H. (2002) "Multi colony ant algorithms," *Journal of Heuristics*, Kluwer Academic Publishers, vol. 8, pp. 305-320.

Miranda G., Luna H. P. L., Mateus G. R. and Ferreira R. P. M. (2005) "A performance guarantee heuristic for electronic components placement problems including thermal effects," *Computers and Operations Research*, vol. 32, pp. 2937-2957.

Misevicius A. (2003) "A modified simulated annealing algorithm for the quadratic assignment problem," *Informatica*, vol. 14, pp.497-514.

Misevicius A. (2005) "A tabu search algorithm for the quadratic assignment problem,"

Computational Optimization and Applications, vol. 30, no. 1, pp. 95-111.

Moscato P. (1989) *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, Caltech Concurrent Computation Program, Report 826, California Institute of Technology, Pasadena, California, USA.

Nguyen Q. H., Ong Y. S. and Lim M. H. (2008) "Non-genetic transmission of memes by diffusion," In: *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, New York, pp. 1017-1024.

Nissen V. (1994) "Solving the quadratic assignment problem with clues from nature," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 66-72.

Nugent C. E., Vollman T. E., and Ruml J. (1968) "An experimental comparison of techniques for the assignment of facilities to locations," *Operations Research*, vol. 16, pp. 150-173.

Ong Y. S. and Keane A. J. (2004) "Meta-Lamarckian in memetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 99-110.

Ong Y. S., Lim M. H. and Chen X. S. (2010) "Research frontier: Memetic computation -past, present & future", *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, pp. 24-36.

Ong Y. S., Lim M. H., Zhu N. and Wong K. W. (2006) "Classification of adaptive memetic algorithms: A comparative study," *IEEE Transactions on Systems, Man, and Cybernetics*, Part B, vol. 36, no. 1, pp. 141-152.

Or I. and Atik K. (2000) "A quadratic assignment problem in the automated assembly of printed circuit boards," *Journal of the Operations Research Society of Turkey*, vol. 11,

pp. 67-88.

Osman I. H. (1993) "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems," *Annals of Operations research*, vol. 41, pp. 421-451.

Poyla G. (1945) *How to solve it*, Princeton University Press, Princeton.

Rabak C. S. and Sichman J. S. (2003) "Using A-Teams to optimize automatic insertion of electronic components," *Advanced Engineering Informatics*, vol. 17, pp. 95-106.

Rechenberg I. (1973) *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog.

Resende M. G. C., Pitsoulis L. S. and Pardalos P. M. (1997) "Approximate solution of weighted MAX-SAT problems using GRASP," In: *Satisfiability Problem: Theory and Applications*, Du D.-Z., Gu J. and Pardalos P. M. (Eds.), AMS: Providence, Rhode Island, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 393-405.

Ribeiro C. C., Souza M. C. and Souza C. (2002) "Variable neighborhood search for the degree-constrained minimum spanning tree problem," *Discrete Applied Mathematics*, vol 118, no 1-2, pp. 43-54.

Rumelhart D. E., Hinton G. E. and Williams R. J. (1986) "Learning internal representation by error propagation," *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, vol. 1, chapter 8, Cambridge, MA, MIT Press.

Sahni S. and Gonzalez T. (1976) "P-complete approximation problems," *Journal of the ACM*, vol. 23, no. 3, pp. 555-565.

Skorin-Kapov J. (1990) "Tabu search applied to the quadratic assignment problem,"

ORSA Journal on Computing, vol. 2, no. 1, pp. 33-45.

Skorin-Kapov J. (1994) "Extensions of a tabu search adaptation to the quadratic assignment problem," *Computers and Operations Research*, vol. 21, no. 8, pp. 855-865.

Smith J. E. (2007) "Coevolving memetic algorithms: A review and progress report," *IEEE Transactions on Systems, Man, and Cybernetics*, Part B, vol. 37, no. 1, pp. 6-17.

Steinberg L. (1961) "The backboard wiring problem: a placement algorithm," *SIAM Review*, vol. 3, pp. 37-50.

Stützle T. (1997) *MAX-MIN ant system for quadratic assignment problems*, Research Report AIDA-97-04, Department of Computer Schience, Darmstadt University of Technology, Germany.

Stützle T. (2006) "Iterated local search for the quadratic assignment problem," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1519-1539.

Stützle T. and Dorigo M. (1999) "ACO algorithms for the quadratic assignment problem," In: *New Ideas in Optimization*, Corne D., Dorigo M., and Glover F. (Eds.), McGraw-Hill, London, pp. 33-50.

Stützle T. and Hoos H. H. (1997) "The MAX-MIN ant system and local search for the traveling salesman problem," In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, Bäck T., Michalewicz Z. and Yao X. (Eds.), IEEE Press, Piscataway, NJ, pp. 309–314.

Taillard E. D. (1991) "Robust taboo search for the quadratic assignment problem,"

Parallel Computing, vol. 17, pp. 443-455.

Taillard E. D. (1995) "Comparison of iterative searches for the quadratic assignment problem," *Location Science*, vol. 3, pp. 87-105.

Taillard E. D. and Gambardella L. M. (1997a) "An ant approach for structured quadratic assignment problems," *Second Metaheuristics International Conference*, Sophia-Antipolis, France.

Taillard E. D. and Gambardella L. M. (1997b) "Adaptive memories for the quadratic assignment problem," *Technical Report IDSIA-87-97*, Lugano, Switzerland, 1997.

Talbi, E. G. (2009) *Metaheuristics: from design to implementation*, Wiley.

Talbi E. G., Roux O., Fonlupt C. and Robillard D. (2001) "Parallel ant colonies for the quadratic assignment problem," *Future Generation Computer Systems*, vol. 17, pp. 441-449.

Tang J., Lim M. H., Ong Y. S. (2007) "Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems," *Soft Computing*, vol.11, pp.873-888.

Tang J., Lim M. H., Ong Y. S. and Er M. J. (2006) "Parallel memetic algorithm with selective local search for large scale quadratic assignment problems," *International Journal of Innovative Computing, Information and Control*, vol. 2, no. 6, pp. 1399-1416.

Tate D. M. and Smith A. E. (1995) "A genetic approach to the quadratic assignment problem," *Computers and Operations Research*, vol. 22, no. 1, pp. 73-83.

Van Batenburg F. H. D., Gultyaev A. P. and Pleij C. W. A. (1995). "An APL-programmed

genetic algorithm for the prediction of RNA secondary structure," *Journal of Theoretical Biology*, vol. 174, no. 3, pp. 269-280.

Van Laarhoven P. J. M., Aarts E. H. L. and Lenstra J. K. (1992) "Job shop scheduling by simulated annealing," *Operations Research*, vol. 40, pp. 113-125.

Vollmann T. E. and Buffa E. S. (1966) "The facilities layout problem in perspective," *Management Science*, vo. 12, no. 10, pp. 450-468.

Voudouris C. and Tsang E. (1999) "Guided local search," European Journal of *Operational Research*, vol. 113, no. 2, pp. 469-499.

Werbos P. J. (1974) "Beyond regression: New tools for prediction and analysis in the behavioural sciences," *Ph.D. Thesis*, Harvard University, Cambridge, MA.

Widow B. and Hoff M. E. (1960) "Adaptive switching circuits," *IRE WESCON Convention Record (1960)*, IRE, New York, Vol. 4, pp. 96-104.

Wilkins J. S. (1998) "What's in a Meme? Reflections from the perspective of the history and philosophy of evolutionary biology," *Journal of Memetics - Evolutionary Models of Information Transmission*, 2. URL: http://cfpm.org/jom-emit/1998/vol2/wilkins_js.html

Wolpert D. H. and Macready W. G. (1997) "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol.1, issue 1, pp.67-82.

Xu Y. L., Lim M. H., Ong Y. S. and Tang J. (2006) "A GA-ACO-Local Search Hybrid Algorithm for Solving Quadratic Assignment Problem," *Genetic and Evolutionary Computation Conference (GECCO'06)*, Seattle, Washington, USA.

Xu Y. L., Lim M. H. and Yan K. H. (2003) "Hybrid-GA with stochastic selection local

search for solving quadratic assignment problems," In: *Proceedings of the second International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2003)*, Special Session on Optimization using Genetic, Evolutionary, Social and Behavioral Algorithms, Singapore.

Yao X. (ed.) (1999) *Evolutionary Computation: Theory and Applications*, World Scientific Publ. Co., Singapore. (ISBN 981-02-2306-4).