A WATER FLOW ALGORITHM FOR OPTIMIZATION PROBLEMS

TRAN TRUNG HIEU

NATIONAL UNIVERSITY OF SINGAPORE

2011

A WATER FLOW ALGORITHM FOR OPTIMIZATION PROBLEMS

TRAN TRUNG HIEU

(B.Eng. (Hons.), HCMUT)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2011

ACKNOWLEDGEMENTS

First of all, I would like to express my sincerest gratitude to my supervisor, Associate Professor Ng Kien Ming, at the Department of Industrial and Systems Engineering, National University of Singapore, for his encouragement and guidance throughout my PhD studying process. His invaluable advices have helped me to successfully complete my research work as well as thesis.

Next, I would like to thank all the lecturers of the Department of Industrial and Systems Engineering, National University of Singapore, for their lessons which helped me to achieve necessary and useful knowledge for my research work. I would also like to extend my acknowledgement to the officers of the department for their assistance in handling my administrative matters.

Finally, I would like to take this chance to express my special gratitude to my beloved girlfriend, Ms. Ry, for her constant love and continuous support throughout my PhD studying process.

Tran Trung Hieu

November 2011

TABLE OF CONTENTS

ABSTRACT

A novel natured-inspired algorithm, called the water flow algorithm (WFA), for solving optimization problems has been proposed in this research work. The proposed algorithm is designed by simulating the hydrological cycle in meteorology and the erosion phenomenon in nature. Basic operators of this algorithm are based on the raindrops distribution simulation, the property of water flow always moving to lower positions, and the erosion process to overcome obstacles. Depending on the structure of a specific problem, the WFA can be appropriately customized to solve the problem efficiently.

In this thesis, we focus on solving well-known combinatorial optimization problems, such as the permutation flow shop scheduling problem (PFSP) in production planning, quadratic assignment problem (QAP) in facility layout design, and vehicle routing problem (VRP) in logistics and supply chain management. The general WFA has been customized and implemented successfully for solving these problems. For the PFSP, the proposed algorithm obtained not only optimal solutions for several PFSP benchmark instances taken from OR Library, but also a new best known solution for the benchmark instance of Heller. For the QAP, the algorithm solved most QAP benchmark instances drawn from the QAPLIB. The comparison results also show that the WFA outperforms other algorithms in terms of solution quality for both the PFSP and the QAP. For the VRP, the WFA is combined with solving the relaxed mathematical programming model of the VRP to search for optimal solutions to this problem. Preliminary results show that this

algorithm is able to obtain optimal solutions for some of the VRP benchmark instances taken from the literature.

Also, the WFA has been developed to solve flexible flow shop scheduling problem (FFSP) with intermediate buffers, which is a general case of the PFSP. The FFSP is more complex than the PFSP since there are a number of parallel identical machines at each stage and intermediate buffers between consecutive stages. To solve the problem, an efficient procedure for constructing a complete schedule is required. Hence, we proposed a procedure for constructing a complete schedule as well as determining corresponding objective values for the FFSP. The procedure is included in the WFA to increase the efficiency of this algorithm. The experimental results and comparisons show that the proposed algorithm outperforms other algorithms in terms of solution quality as well as computation time. Moreover, the WFA has obtained new upper bound solutions for several Wittrock benchmark instances.

The WFA can also be modified to solve multi-objective optimization problems. In this thesis, we have designed the WFA for solving multi-objective scheduling problems, called the MOWFA. Landscape analysis as well as evaporation and precipitation processes are integrated into the MOWFA to solve the multi-objective FFSP with limited buffers efficiently. Experimental results show that the MOWFA outperforms an improved hybrid multi-objective parallel genetic algorithm for the multi-objective scheduling problem.

In conclusion, the WFA is able to obtain optimal or good quality solutions to several well-known combinatorial optimization problems within reasonable computation time. It is thus a promising algorithm to solve other types of optimization problems as well.

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

WFA	Water flow algorithm
MOWFA	Multi-objective water flow algorithm
2LWFA	Two-level water flow algorithm
PFSP	Permutation flow shop scheduling problem
FFSP	Flexible flow shop scheduling problem
MOFFSP	Multi-objective flexible flow shop scheduling problem
QAP	Quadratic assignment problem
VRP	Vehicle routing problem
CVRP	Capacitated vehicle routing problem
P_0-list	The best position list
UE-list	Un-eroded position list
E-list	Eroded position list
DOW	Drop of water
MaxCloud	The maximum number of clouds generated
MaxPop	The maximum number of *DOW*s generated in each cloud
MinEro	The minimum amount of precipitation allowed to star erosion process
MaxUIE	The maximum number of iterations for erosion process

CHAPTER 1

INTRODUCTION

Combinatorial optimization problems are considered as a category of general optimization problems in which the values of decision variables are subjected to integrality constraints. Such problems are commonly encountered in the real world, especially in the field of industrial systems (Yu, 1998). Because of the integrality constraints, their set of feasible solutions is finite and an enumeration method may be applied to find an optimal solution for these problems. However, the solution search space of combinatorial optimization problems can grow exponentially according to the size of the problems, and many of them are classified as NP-hard problems (Papadimitriou and Steiglitz, 1982). Thus when solving combinatorial optimization problems with large size, exact optimization methods, such as the branch and bound method, may not obtain optimal solutions to these problems within acceptable computation time; even heuristic algorithms may face difficulties finding good quality solutions. Recently, meta-heuristics, especially nature-inspired algorithms, have achieved some success with solving certain combinatorial optimization problems in reasonable computation time (Shah-Hosseini, 2007; and Yang, 2008). The results show the potential of such approaches in solving other types of combinatorial optimization problems.

In this thesis, we propose a novel nature-inspired meta-heuristic algorithm that is able to solve different types of combinatorial optimization problems. A brief description of such problems that can be solved by the proposed algorithm is provided in the next section. We also present an overview of the practical applications of combinatorial optimization problems in this section. As meta-heuristic algorithms are able to obtain solutions for certain combinatorial optimization problems effectively, we describe the features of meta-heuristic algorithms, especially nature-inspired ones, as well as give a classification of the algorithms in Section 1.2. Next, we present the motivation and research objectives of the thesis in Section 1.3. Then, the main contributions of the thesis are summarized in Section 1.4. Finally, the organization of this thesis is provided in Section 1.5.

1.1 Combinatorial Optimization Problems

Many real-world optimization problems can be formulated as mathematical programming models with integrality constraints. The modeling and solving of such real-world problems are related to "Combinatorial Optimization" (Christofides et al., 1979). In a combinatorial optimization problem, the set of feasible solutions is discrete, or can be reduced to a discrete set. Some examples of classical combinatorial optimization problems consist of the knapsack problem that arises in resource allocation with financial constraints or electronic transfer of funds (Martello and Toth, 1990), traveling salesman problem that is applied to product distribution or the production of printed circuit boards (Applegate et al., 1994), and the multi-commodity flow problem that has applications in production planning or warehousing (Ahuja et al., 1993). In addition, other well-known examples of combinatorial optimization problems in production and logistics include the

permutation flow shop scheduling problem, flexible flow shop scheduling problem, quadratic assignment problem and vehicle routing problem, which are described below:

1. **Permutation Flow Shop Scheduling Problem**: This problem involves determining a sequence of *n* jobs to be processed through *m* machines with the same order of jobs on the machines. Because of the same order of jobs through all machines, the sequence of jobs processed on the first machine is considered as a feasible solution of the problem. The most popular objective of this problem is to minimize the completion time of jobs, also known as makespan (C_{max}). An integer programming model for the scheduling problem was presented in Pinedo (2005).

2. **Flexible Flow Shop Scheduling Problem**: This problem involves a set of jobs processed through several consecutive operation stages with parallel identical machines in each stage. A job can be processed on any idle machine at the stage into which the job is going. In some cases, there are limited intermediate buffers between consecutive stages. The primary objective of this problem is to find a production schedule to minimize the completion time of jobs. There are also other important objectives of this problem, such as to minimize the total weighted flow time of jobs and to minimize the total weighted tardiness time of jobs. This problem can be formulated as a mixed-integer programming model (Sawik, 2000).

3. **Quadratic Assignment Problem**: This problem is first stated by Koopmans and Beckmann (1957). Given a set of facilities and a set of locations with the same size *n*, the aim is to assign the facilities to the locations such that the total cost is minimized. The total cost *w* is calculated using the distance between locations and

the flow between facilities. This problem can then be expressed as the problem of finding a permutation π of n facilities as follows:

$$\min_{\pi \in \Pi_n} w(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}\, d_{\pi[i]\pi[j]}, \tag{1.1}$$

where Π_n denotes the set of possible permutations of $N = \{1, 2, \ldots, n\}$, while $\pi[i]$ and $\pi[j]$ denote the location of facilities i and j in the permutation π respectively. Furthermore, f_{ij} is the flow between facilities i and j, and $d_{\pi[i]\pi[j]}$ is the distance between locations $\pi[i]$ and $\pi[j]$.

4. **Vehicle Routing Problem**: The problem on how to service a number of customers' demand with a fleet of given vehicles under some specific constraints is known as a vehicle routing problem. Such a problem was first proposed by Dantzig and Ramser (1959), and it is currently well-known with many important applications in the field of logistics and supply chain management. An objective of the vehicle routing problem is to assign the vehicles to the customers and find the efficient routes of the vehicles so that the total travel distance or time is minimized. This problem may be formulated as an integer programming model.

1.2 Nature Inspired Algorithms

In this section, we show a classification of optimization methods for solving combinatorial optimization problems in Figure 1.1. The classification is based on the operational mechanism of optimization methods. Our focus is on nature-inspired algorithms, which

can be considered as a type of meta-heuristic algorithms. Here the term "meta-" means "beyond" or "higher level", while the term "heuristic" means "search" or "discover by trial or error" (Yang, 2008). Thus, meta-heuristic algorithms are known as optimization methods that search for optimal solutions of a problem by iteratively improving a candidate solution with regards to a given objective function.

Meta-heuristic algorithms usually consist of two major processes, i.e., solution exploration and solution exploitation. These two processes are iteratively performed to search for optimal or near-optimal solutions in reasonable computation time. The exploration process not only increases the diversity of solutions found, but also helps to overcome local optimal solutions to obtain better or optimal ones due to its randomization. The exploitation process aims to improve the quality of solutions obtained from the exploration process in order to ensure that the solutions will converge to optimality. In some meta-heuristic algorithms, this exploitation process also helps to overcome local optimal solutions to search for better or optimal ones. The performance of meta-heuristic algorithms depends on the appropriate combination between these two processes. Because of the features of meta-heuristic algorithms, they can search for solutions of combinatorial optimization problems with good quality in realistic computation time. Some well-known applications can be found in Liao et al. (2007) and Yang (2008).

We have classified meta-heuristic algorithms into two major types, i.e., nature-inspired algorithms and non-nature inspired algorithms. Nature has been evolving for millions of years and hence learning from the success of nature to design meta-heuristic algorithms is a creative idea (Yang, 2008). Nature-inspired algorithms can be further divided into biologically inspired algorithms, botanically inspired algorithms and water flow inspired

Figure 1.1 A Classification of Algorithms for Combinatorial Optimization Problems

techniques. A detailed description of these types of nature-inspired algorithms is given in Chapter 2. We also present the important applications of these algorithms in the same chapter.

1.3 Motivation and Research Objectives

Real-world problems that are formulated as combinatorial optimization problems often face the following difficulties: (a) their problem size or dimension is large, (b) they are computationally complex to solve, and (c) even after decomposing into simpler sub-problems, they are still NP-hard problems. Although the search space of such problems is determined by a finite set of feasible solutions, it grows exponentially with the size of the problems. Hence, it is sometimes sufficient to obtain a near-optimal solution to such problems in practice. As such, there is a need for constructing good meta-heuristic algorithms that can search for solutions with good quality in realistic computation time.

Over the last few decades, meta-heuristic algorithms inspired by natural phenomena have been extensively developed to become search and optimization tools in various optimization problems. Based on the inherent features of meta-heuristic algorithms, nature-inspired algorithms have been successful in solving many optimization problems. However, the algorithms are only efficient on specific problems, and there is a need to change their operational mechanism to solve other problems. There is thus a lack of such algorithms that are able to solve diverse combinatorial optimization problems.

The success of nature-inspired algorithms for solving optimization problems has motivated us to learn about their potential capability in constructing an algorithm inspired

by other natural phenomena. Thus the main research objective of this thesis is to design a novel nature-inspired algorithm for solving combinatorial optimization problems. Also, the algorithm has to balance between solution exploration and exploitation capabilities to achieve optimal solutions in realistic computation time to real-world problems. Moreover, this thesis aims to develop a nature-inspired algorithm that can solve a variety of optimization problems with similar problem structure. Another objective of the thesis is to design the algorithm in such a way that it is able to solve both single-objective and multi-objective optimization problems efficiently. Finally, the algorithm should not have difficulty hybridizing with other well-known algorithms to increase the algorithm's solution efficiency.

One of the well-known natural phenomena is the hydrological cycle in meteorology and the erosion process of water flow in nature. They possess features that are suitable for developing an efficient optimization algorithm. As such, this thesis is focused on utilizing such natural phenomena in developing a novel nature-inspired algorithm, and then testing the performance of the resulting algorithm with different types of NP-hard combinatorial optimization problems.

1.4 Main Contributions of the Thesis

In this section, we summarize the main contributions of this thesis as follows:

Firstly, a novel nature-inspired algorithm, known as the water flow algorithm (WFA), for solving NP-hard optimization problems is proposed. The proposed algorithm has mostly imitated the characteristics of water flow in nature and the components of the

hydrological cycle in meteorology. This algorithm consists of two major phases, i.e., the solution exploration and exploitation phases, inspired by the hydrological cycle and the erosion phenomenon, respectively.

Secondly, the WFA has been developed to successfully solve several NP-hard optimization problems, such as permutation flow shop scheduling problem (PFSP), flexible flow shop scheduling problem (FFSP) with intermediate buffers, quadratic assignment problem (QAP), and capacitated vehicle routing problem (CVRP). Almost all the best known solutions of the benchmark instances used can be found by the proposed algorithm. Moreover, this algorithm can obtain many new best known solutions for PFSP and FFSP.

Thirdly, we have constructed a WFA which is able to solve multi-objective optimization problems by modifying and integrating some specialized components. It was applied to solve the multi-objective FFSP with intermediate buffers. In this algorithm, landscape analysis based on the ellipsoid approximation was integrated to help determine the weights of objective functions, which guide the WFA to exploit potential regions and move towards the optimal Pareto solution set. The results obtained demonstrate the effectiveness and efficiency of the WFA, and show that this is a promising approach to solve other multi-objective optimization problems.

In addition, when WFA is developed to solve the FFSP with intermediate buffers, we also propose an efficient procedure of constructing a complete schedule from a job permutation at the first stage. The constructive procedure outperforms other procedures in the literature. This procedure may increase the performance of any algorithm when it is

integrated to solve the FFSP. Also for solving the FFSP, the WFA is applied to the maltose syrup production problem. The results obtained show the capability of the WFA for solving problems in complex real-world applications.

Lastly, a constructive algorithm for generating initial solutions of the WFA when solving the CVRP is proposed in this thesis. The constructive algorithm is based on solving a relaxed CVRP. The results obtained show that this algorithm may generate good initial solutions for the CVRP. Furthermore, this approach of finding initial solutions may be applied to other optimization problems.

1.5 Outline of the Thesis

This thesis aims to develop a novel nature-inspired algorithm for combinatorial optimization problems and its content is organized into eight chapters. Figure 1.2 shows the organization and relationship among the chapters.

A detailed review of nature-inspired algorithms that have emerged in recent years is provided in Chapter 2. There are three main groups in the literature review corresponding to the classification of nature-inspired algorithms shown in Section 1.2. The basic ideas of designing nature-inspired algorithms, the development of the algorithms, as well as their successful applications are also described in this chapter.

Chapter 3 describes the phenomena of nature used to construct the WFA, i.e., the hydrological cycle in meteorology and the erosion process of water flow in nature. Also, the operational mechanism and the main operators of the proposed algorithm are explained in detail.

Chapters 4 and 5 describe the implementation of WFA for two single-objective combinatorial optimization problems, namely the permutation flow shop scheduling problem and the flexible flow shop scheduling problem, respectively. Computational results and comparisons carried out on benchmark problems are also presented and discussed. In addition, a practical example of maltose syrup production solved by the WFA is shown in Chapter 5. Chapter 6 presents on how the WFA can be developed to solve a multi-objective flexible flow shop scheduling problem, and computational results are also shown in this chapter.

Some further applications of the WFA for other single-objective combinatorial optimization problems, such as the quadratic assignment problem and vehicle routing problem, are described in Chapter 7. The proposed algorithms are also tested with the benchmark instances from the literature, with the computational results and comparisons being shown in this chapter. Finally, some conclusions are provided in Chapter 8. The contributions of this research work are also discussed, together with some possible future research works.

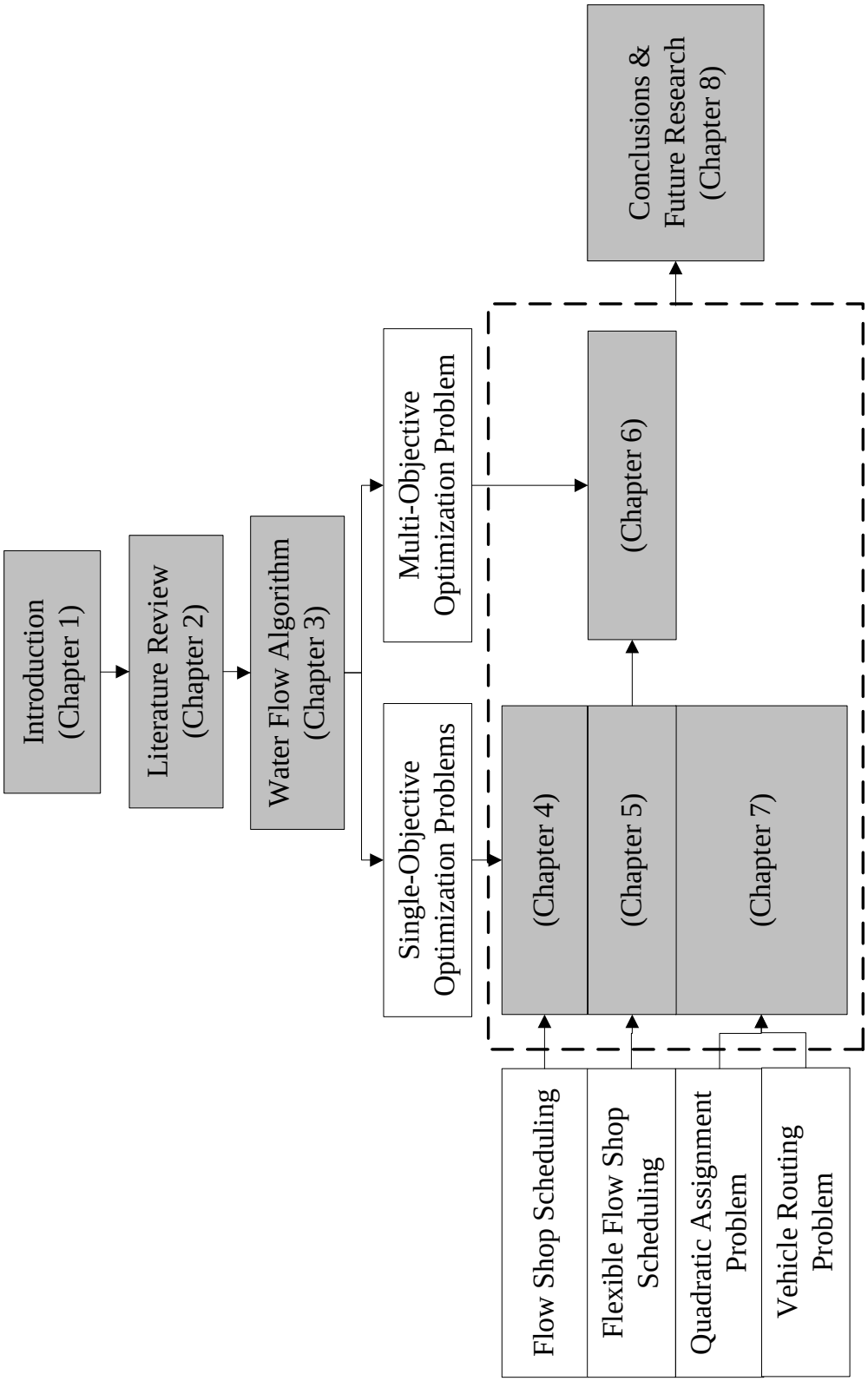


Figure 1.2 Organization of the Thesis

CHAPTER 2

LITERATURE REVIEW

Nature-inspired algorithms have become useful optimization methods for solving a variety of real-world problems. Based on the appropriate balance between solution exploration and exploitation capabilities, the algorithms can obtain solutions with high quality in realistic computation time. In this chapter, we present several well-known nature-inspired algorithms in recent years. The basic ideas of constructing and developing the algorithms, as well as their most important applications are described in detail. Here, the nature-inspired algorithms are classified into three main groups: biologically inspired algorithms, botanically inspired algorithms and water flow inspired techniques. A survey of these three groups of algorithms is presented in Section 2.1, Section 2.2 and Section 2.3 respectively. Also, some findings from the literature review are presented in this section. Finally, a timeline of all the nature-inspired algorithms and the additional features of the proposed nature-inspired algorithm which help to overcome the drawbacks of existing algorithms are shown.

2.1 Biologically Inspired Algorithms

In this section, we present some popular meta-heuristic algorithms inspired from biology. The biologically inspired algorithms are classified into three major groups: evolutionary algorithms, stigmergic optimization algorithms, and swarm-based optimization algorithms. The group of evolutionary algorithms consists of genetic algorithm, memetic algorithm, and shuffled frog-leaping algorithm. The group of stigmergic optimization algorithms includes termite algorithm, ant colony optimization and bee colony optimization. Finally, the group of swarm-based optimization algorithms includes particle swarm optimization, firefly algorithm, and bat algorithm.

2.1.1 Evolutionary Algorithms

Evolutionary algorithms are stochastic optimization methods based on the principles of natural evolution. Natural evolution is a complex process which operates on chromosomes, instead of organisms (Michalewicz, 1992). The chromosomes contain genetic information, called a gene, which is passed from one generation to next generation through reproduction. In reproduction, the most important operators are recombination and mutation. The recombination plays a role in the exchange of genetic information among parent individuals to produce an offspring, while the mutation aims to create diversification of genetic information in offspring. Organisms with good chromosomes have a higher chance to exist and develop in nature. According to Darwin's natural selection theory (Darwin, 1859), natural selection prioritizes the proliferation of environment-adapted organisms, but causes the extinction of non-environment-adapted organisms.

A well-known evolutionary algorithm is genetic algorithm (GA) first introduced by Holland (1975). The idea of establishing GA originated from the evolutionism of Darwin (1859): "*Survival of the genetically fittest*". Since the introduction of this algorithm, it has become a popular and useful optimization method for solving optimization problems. Although many variants of GA have been developed, the general framework of this algorithm does not have any significant change. The basic principles of GA are described in detail by Goldberg (1989). A flow chart of the general GA is shown in Figure 2.1.

The solution exploration capability of GA increases because of the initial population. However, a certain degree of exploiting the regions with high quality solutions is missing. Hence, the GA is combined with a local search to overcome the drawback, which constitutes a memetic algorithm (MA). In particular, the MA allows all chromosomes as well as individuals to gain some experience through a local search process before they are evolved. The MA was first introduced by Moscato (1989). In this algorithm, the genetic information to form a chromosome is called memes and not genes. This is inspired by Dawkins' notion of a meme (Dawkins, 1976). A detailed description of the algorithm can be found in Moscato and Cotta (2003). Here, a flow chart of the MA is shown in Figure 2.2.

Eusuff and Lansey (2003) proposed a shuffled frog-leaping algorithm which combines the advantages of MA and the social behavior of frogs. In this algorithm, a set of frogs similar to a population of individuals in GA is partitioned into subsets, called memeplexes. The memeplexes representing the different cultures of frogs are improved through a process of memetic evolution. Based on the social behavior of frogs, the good evolved knowledge obtained is passed among memeplexes to help the memeplexes evolve together.

The process continues to be performed until stopping criteria are satisfied. A flow chart of the shuffled frog-leaping algorithm is shown in Figure 2.3.

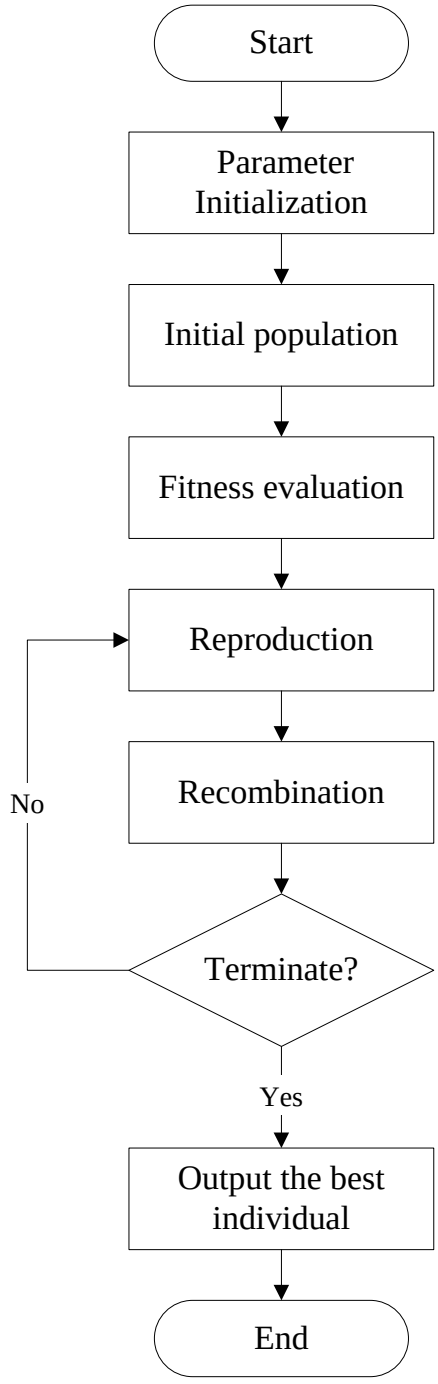


Figure 2.1 Flow Chart of Genetic Algorithm

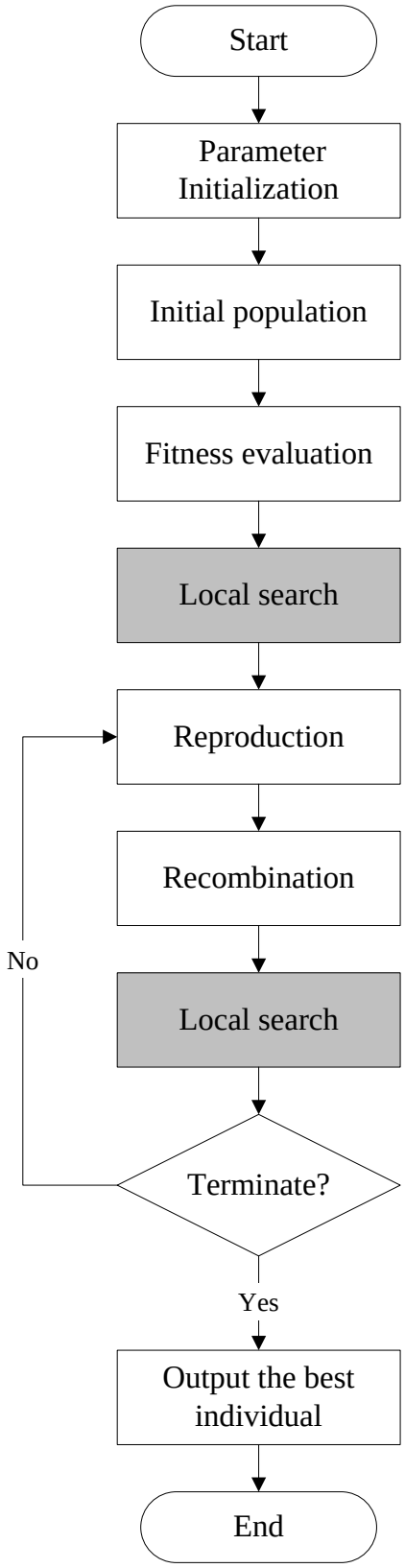


Figure 2.2 Flow Chart of Memetic Algorithm

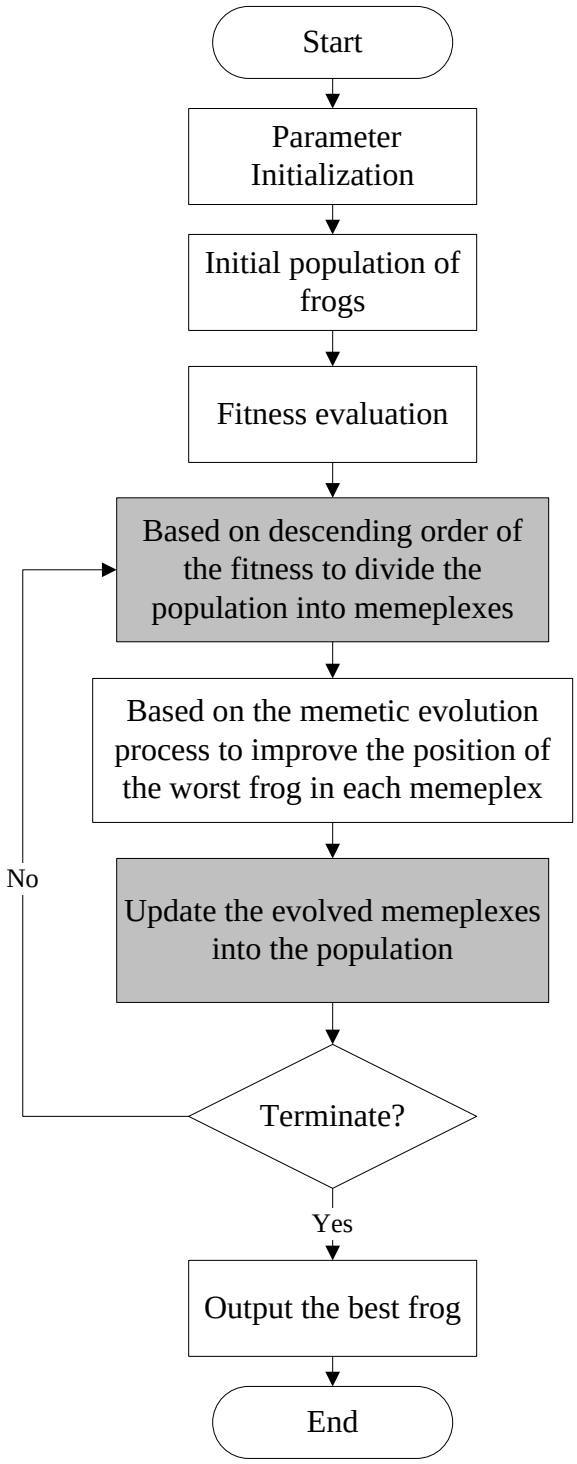


Figure 2.3 Flow Chart of Shuffled Frog-Leaping Algorithm

2.1.2 Stigmergic Optimization Algorithms

According to (Abraham et al., 2006) on stigmergic optimization, Grasse´ quoted: "*Self-Organization in social insects often requires interactions among insects: such interactions can be direct or indirect. Direct interactions are the "obvious" interactions: antennation, trophallaxis (food or liquid exchange), mandibular contact, visual contact, chemical contact (the odor of nearby nestmates), etc. Indirect interactions are more subtle: two individuals interact indirectly when one of them modifies the environment and the other responds to the new environment at a later time. Such an interaction is an example of stigmergy*".

We can observe such an indirect interaction from social insects, such as termites. In the process of nest reconstruction, termites interact through local pheromone concentrations on the nest structure. The state of nest structure coordinates tasks for termites. They work together until the nest construction is completed. Another example of stigmergy is pheromone communication in ant colony. Ants are able to leave a chemical trail on their path to guide other ants to the food source found. Deneubourg et al. (1987) carried out experimental studies to test the ability of ants in searching for the shortest path to the food source. Similar to ants, honey bees can communicate by pheromones. They can deliver a chemical message to encourage attack response to other bees. In addition, honey bees can communicate by "waggle dances". The so-called waggle dances play a role as a signal system which is used to guide other bees to the path to a good food source. Seeley et al. (1991) carried out experimental studies about the ability of bees in allocating and collecting their flower patches.

Many meta-heuristic algorithms can be constructed by learning the behavior of these social insects. We present well-known nature-inspired algorithms in this stigmergic optimization group, such as termite algorithm, ant colony optimization and bee colony optimization.

The first algorithm considered in this group is the termite algorithm proposed by Roth and Wicker (2006) for mobile wireless ad-hoc networks. The proposed algorithm is inspired by the hill building behavior of termites with four principles. Resnick (1997) described the principles as well as a detailed example of the hill building procedure. In the example, it is assumed that termites and pebbles are distributed on a flat surface. Since the hill of termites is built from the pebbles, the objective of termites is to collect all the pebbles into the same place. Termites move based on pheromone trails which are excreted by the others. Following the trails, termites complete their hill building together. To achieve success, a termite must conform to the rules described in Roth and Wicker (2006). A detailed description of matching the hill building procedure of termites and the principles of swarm intelligence to design the termite algorithm is also provided in Roth and Wicker (2006).

The second algorithm considered in this group is the ant colony optimization (ACO) inspired by the foraging procedure of real ants in nature. Although an ant is very tiny and wanders aimlessly, a colony of ants expresses an extraordinarily intelligent behavior through their nest building and foraging. ACO was first introduced by Dorigo and his colleagues around 1991-1992. Since then, it is known as a useful nature-inspired algorithm for combinatorial optimization problems (Dorigo and Gambardella, 1997; and Dorigo, 1999). In the literature, there is a variety of ACO algorithms. However, all of

them have the same framework that is described in Grosan and Abraham (2006) in detail. Here, a flow chart of the basic ACO algorithm is shown in Figure 2.4.

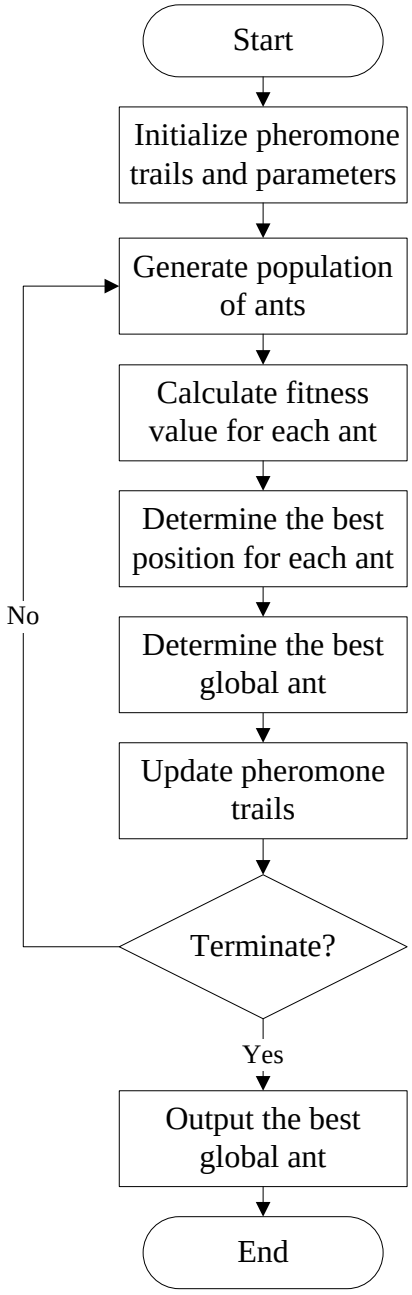


Figure 2.4 Flow Chart of Ant Colony Optimization Algorithm

The third algorithm considered in this stigmergic optimization group is the bee algorithm. The bee algorithm is inspired by the foraging behavior of honey bees. The honey bee algorithm proposed by Nakrani and Tovey (2004) for allocating computers among different clients and web hosting server is one of the earliest bee algorithms. The idea of constructing this algorithm has originated from how forager bees can optimally collect an amount of nectar if they are allocated to different flower patches. On the other hand, Haddad et al. (2005) developed a honey-bee mating optimization algorithm for solving a reservoir operation problem. This algorithm originated from the behavior of queen bee in mating with other bees to form the bee colony. Farooq (2006) presented a bee algorithm for routing in telecommunication network. The algorithm is inspired by the way bees communicate.

Although various bee algorithms have been developed based on the different behavior of honey bees in foraging and mating, they still keep to the same framework. A detailed survey of bee colony algorithms is presented in Bitam et al. (2010). A flow chart of the basic bee algorithm is shown in Figure 2.5.

2.1.3 Swarm-Based Optimization Algorithms

In this section, we present a group of swarm-based optimization algorithms. The algorithms are inspired by the social behavior of swarm-based animals or insects, especially those in which the property of historical information exchange among individuals is magnified. The well-known algorithms in this group include particle swarm optimization, firefly algorithm and bat algorithm.

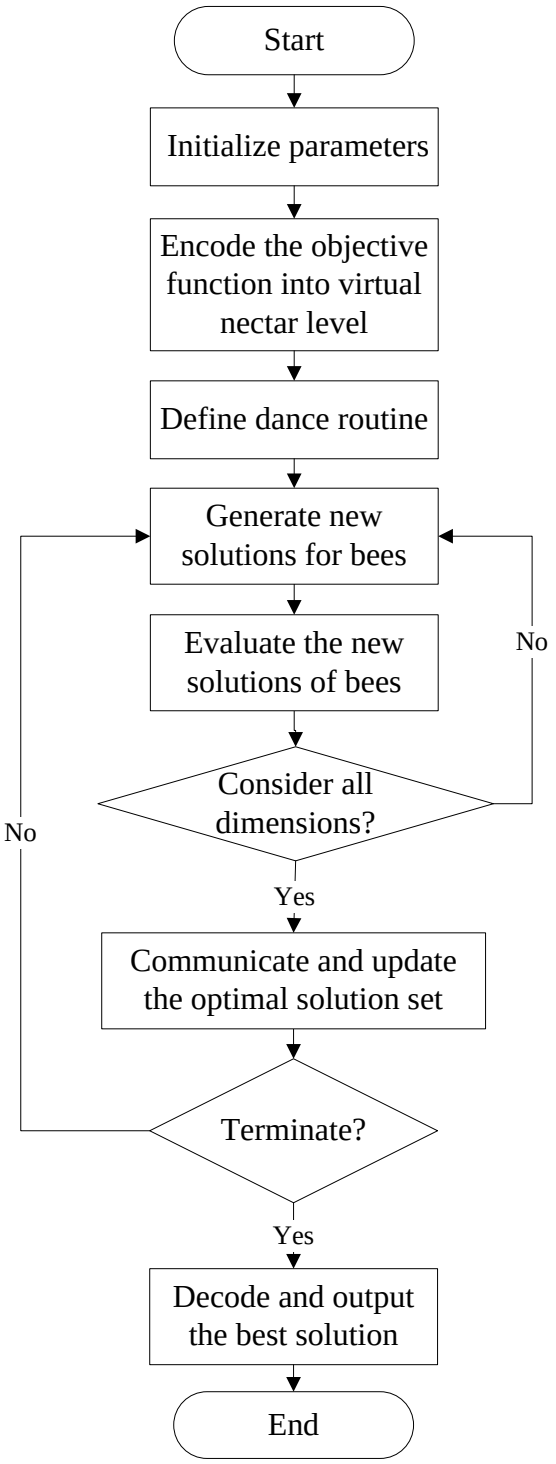


Figure 2.5 Flow Chart of Bee Colony Optimization Algorithm

The first algorithm considered in this group is particle swarm optimization (PSO). This is a biologically inspired algorithm that simulates the social behavior of animals, such as a school of fish or a flock of birds. Here, the basic idea of this algorithm is to use many autonomous agents (particles) that act together in simple ways to produce seemingly complex behavior (Banks et al., 2007). The PSO was first formally introduced by Kennedy and Eberhart (1995). In the paper, the authors replaced the simple goal of the model proposed by Reynold (1987) with a more realistic goal to find food. With this new goal, they established a general model for PSO which is used by most researchers in PSO. Since PSO was developed from an original flocking system, various components were incorporated into the algorithm, such as inertia and constriction. Also, some unnecessary components were removed, such as velocity matching and collision avoidance. However, the standard structure of PSO has generally been preserved. A detailed description of the standard PSO structure is provided in Kennedy and Eberhart (1995). A flow chart of the standard PSO algorithm is shown in Figure 2.6.

The second algorithm considered in this group is the firefly algorithm introduced by Yang (2008). This algorithm is a swarm-based optimization method for solving continuous optimization problem. The algorithm was motivated by simulating the social behavior of fireflies. The fireflies produce short and rhythmic flashes to attract mating partners and potential prey. Also, such flashes of the fireflies may be used to warn potential predators about their bitter taste. A signal system including the rhythmic flash, the rate of flashing and the amount of flashing time attracts both genders together. If female fireflies are attracted by a male firefly, they will respond to the male firefly's pattern of flashing. A flow chart of the algorithm is shown in Figure 2.7.

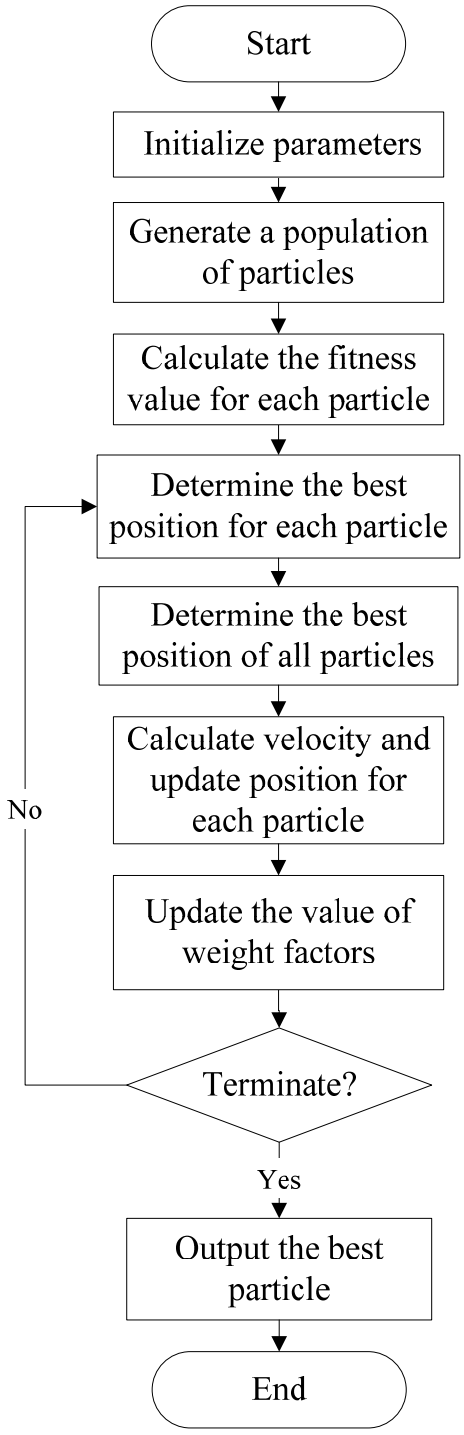


Figure 2.6 Flow Chart of Particle Swarm Optimization Algorithm

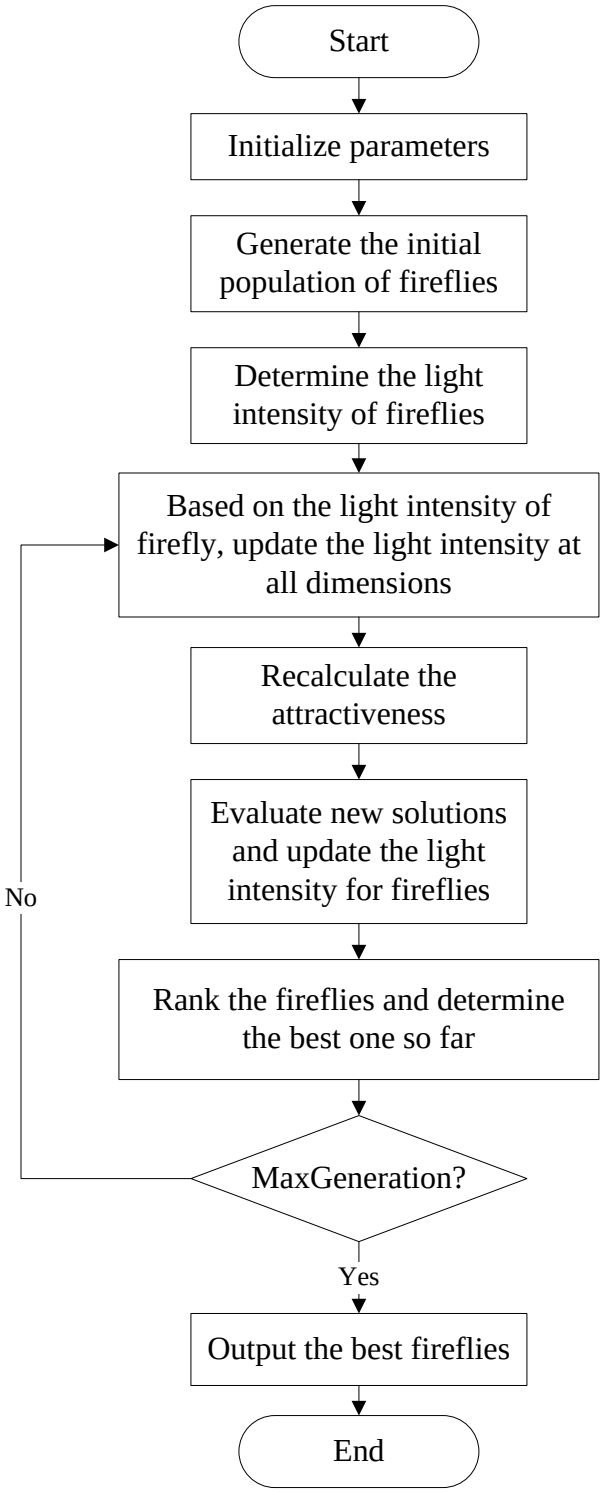


Figure 2.7 Flow Chart of Firefly Algorithm

The third algorithm considered in the group of swarm-based optimization algorithm is the bat algorithm proposed by Yang (2010). The algorithm is an optimization method based on the echolocation behavior of bats. Among all the species of bats, the behavior of microbats motivated the bat algorithm, since they use echolocation extensively, unlike other bats (Richardson, 2008). Microbats' echolocation capability helps them to detect preys, distinguish different kinds of insects, avoid obstacles, and locate their roosting crevices in the dark. A detailed description of the bat algorithm is provided in Yang (2010). Here, a flow chart of the algorithm is shown in Figure 2.8.

2.2 Botanically Inspired Algorithms

In this section, we describe two meta-heuristic algorithms inspired from botany. They consist of an invasive weed optimization algorithm and a grafting-inspired algorithm.

2.2.1 An Invasive Weed Optimization Algorithm

Invasive weed optimization (IWO) is a numerical stochastic optimization algorithm inspired by the ecological process of weed colonization and distribution. In biology, weeds are plants whose invasive habits of growth are vigorous. They threaten the growth of cultivated plants, which in turn poses a threat to agriculture. Weeds are also found to be very robust and adaptive to changes of environment. Hence, Mehrabian and Lucas (2006) used their prominent properties, such as robustness, adaptation and randomness, to design a simple but efficient optimization algorithm for real parameter optimization. A detailed description of the algorithm is provided in Mehrabian and Lucas (2006). A flow chart of the IWO is shown in Figure 2.9.

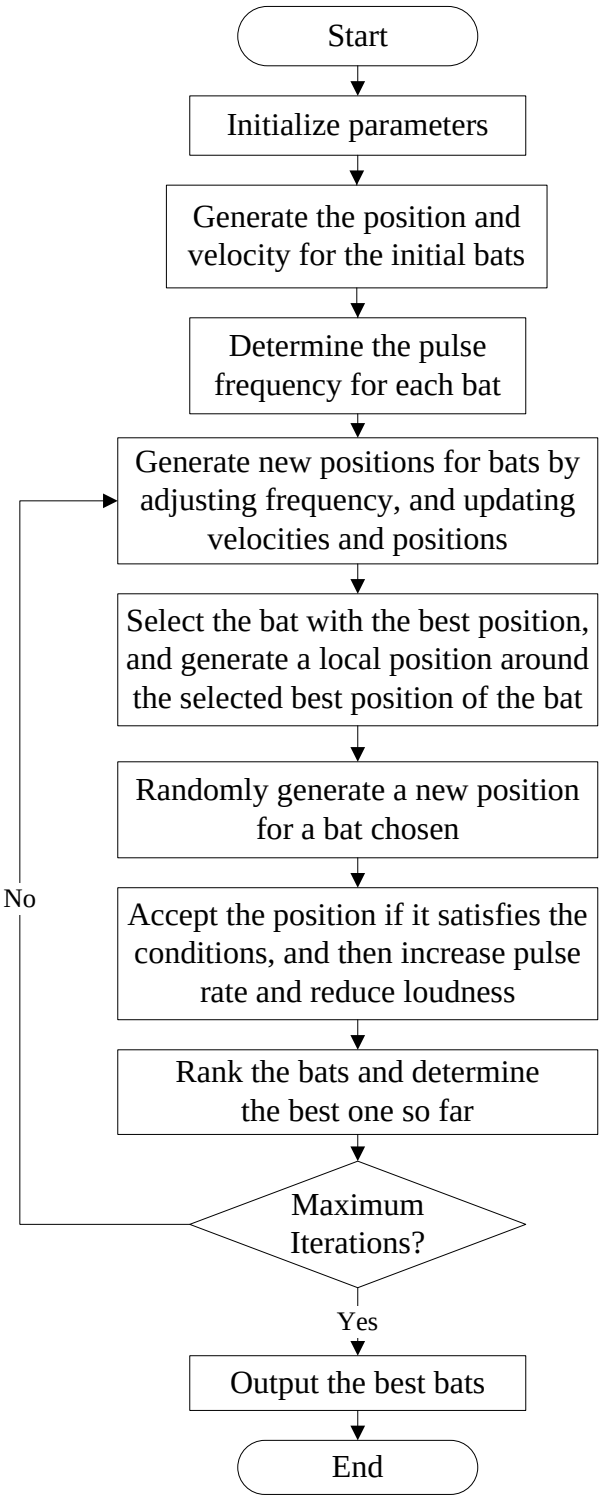


Figure 2.8 Flow Chart of Bat Algorithm

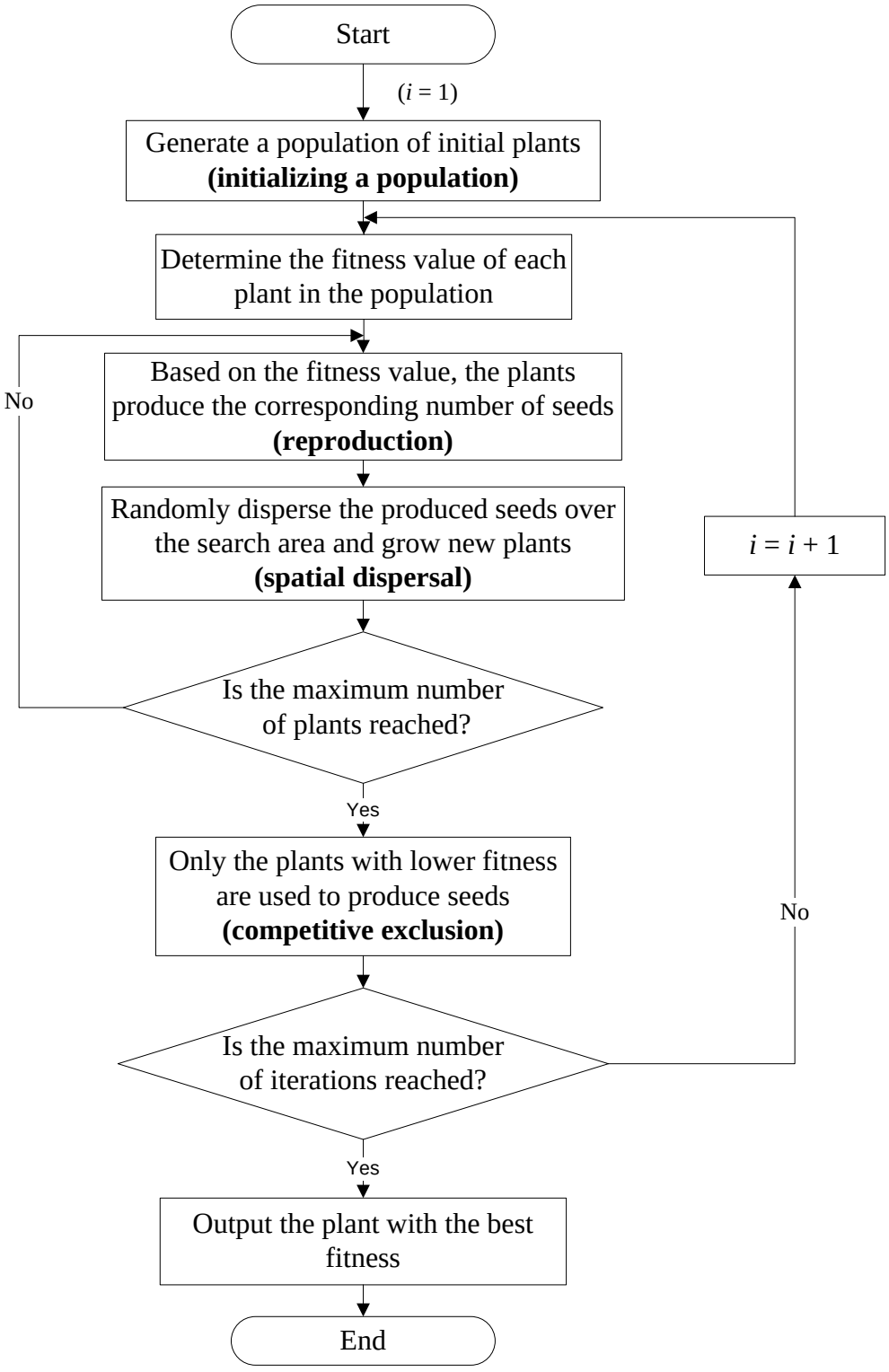


Figure 2.9 Flow Chart of Invasive Weed Optimization

2.2.2 A Botany-Grafting Inspired Algorithm

Li et al. (2010) proposed a hybrid algorithm that combines an improved genetic algorithm (IGA) and an improved particle swarm optimization (IPSO), called HIGAPSO. The combination is inspired by the idea of grafting in botany. Based on the principle that grafting in botany can integrate the strengths of two original branches, the hybrid algorithm combines the advantages of IGA and IPSO together. A flow chart of the proposed hybrid algorithm is shown in Figure 2.10.

2.3 Water Flow Inspired Techniques

In this section, we introduce some recent techniques inspired by certain factors or processes of nature that are related to water flow. The techniques consist of document image processing methods based on water flow model, intelligent water drops algorithm based on the dynamics of river systems, and water flow-like algorithm based on the behavior of fluid flows. These techniques have been successfully applied to a variety of optimization areas, such as pattern recognition, manufacturing and logistics.

2.3.1 Image Processing Methods Based on Water Flow Model

Kim et al. (2002) can be considered as one of the earliest authors who used the properties of water flow to solve optimization problems. In the paper, the authors used a water flow model to simulate an image processing problem. In particular, a grey level image is considered as a three dimensional terrain including mountains and valleys, which represent background and character regions in the image processing problem, respectively.

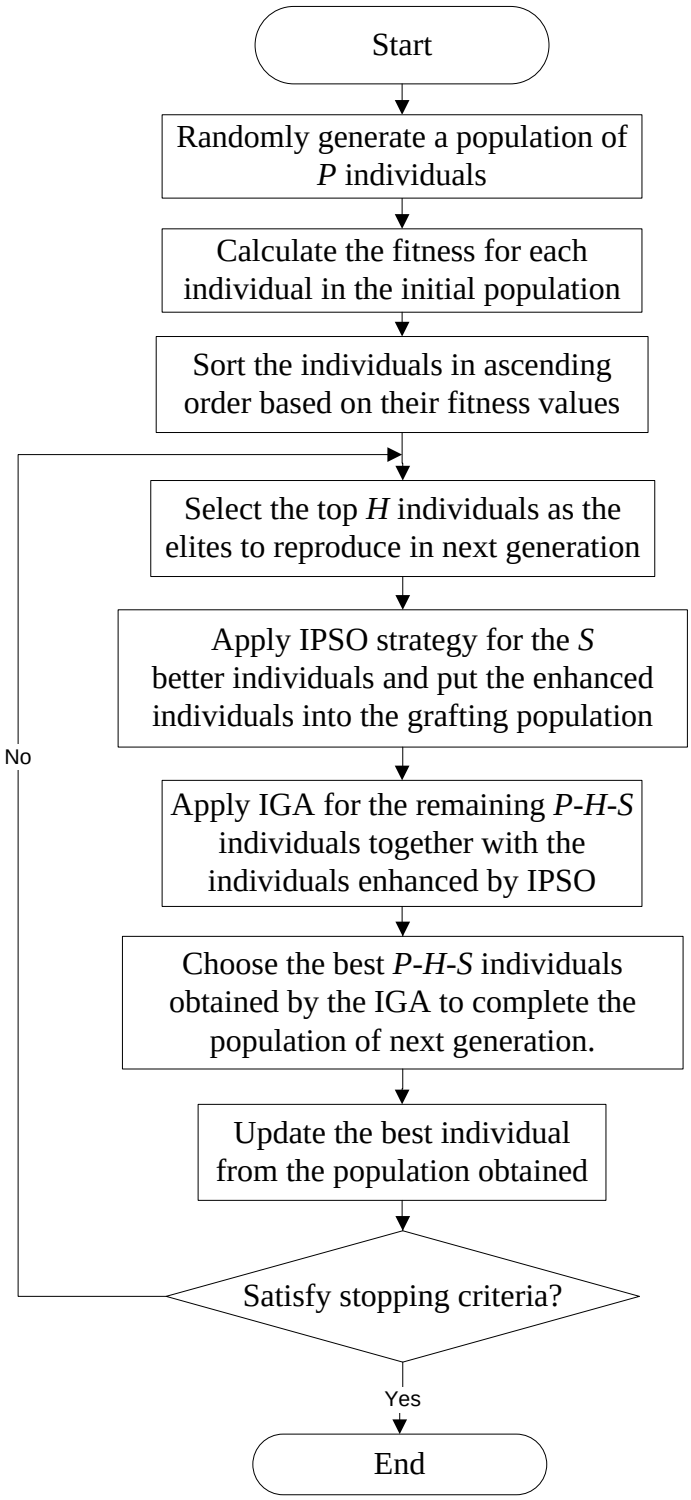


Figure 2.10 Flow Chart of Botany-Grafting Inspired Algorithm

Based on the property of water flow always moving to lower regions, the authors simulated with the pouring of water onto the terrain surface. In the field of image processing, the objective often considered is to extract characters from backgrounds. Thus, a threshold process is proposed to extract the valleys by the amount of filled water. A flow chart of the whole method of Kim et al. (2002) is shown in Figure 2.11.

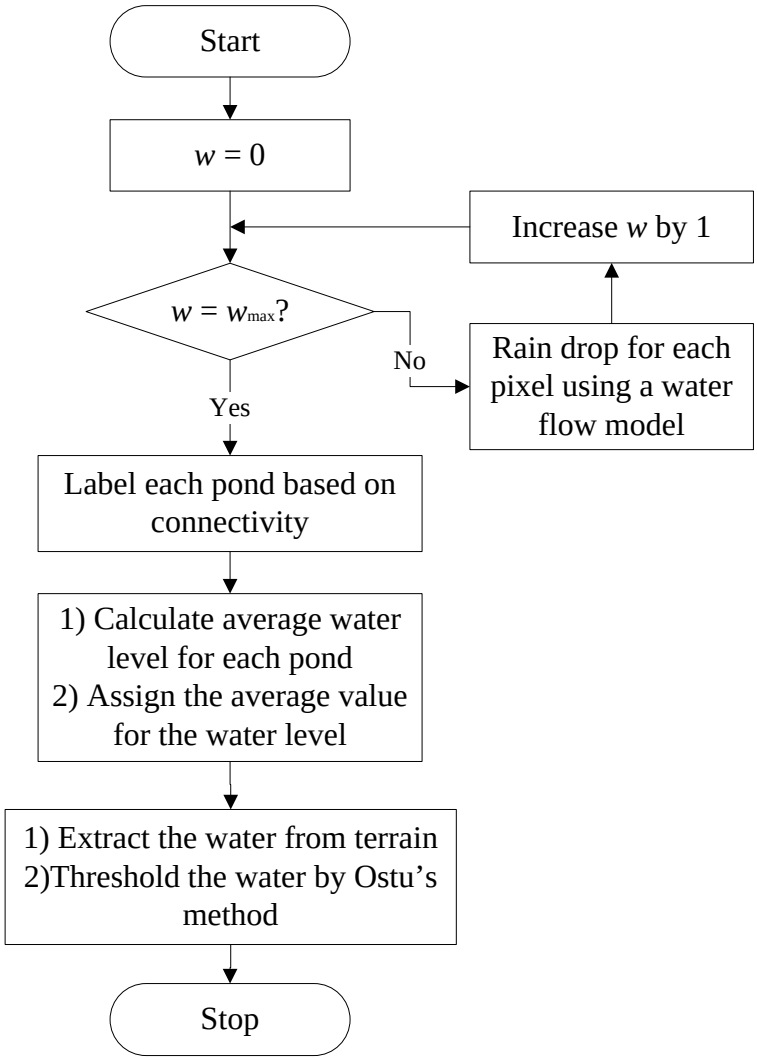


Figure 2.11 Flow Chart of the Method Proposed by Kim et al. (2002)

However, the method of Kim et al. (2002) requires long processing time since it does not determine when to stop the iterative process. Also, characters on poor contrast backgrounds in document images are often not separated successfully. Hence, Oh et al. (2005) proposed an improved method which includes extraction of regions of interest (ROI), an automatic stopping criterion, and hierarchical threshold process, to overcome the drawbacks in the method of Kim et al. (2002). In the improved method, the input image, known as terrain, is divided into regions of interest and desert regions. Then, rainfall only occurs within the regions of interest. An automatic terminating criterion for the iterative rainfall process was then provided to reduce the computational time needed by the model. A flow chart of the method of Oh et al. (2005) is shown in Figure 2.12.

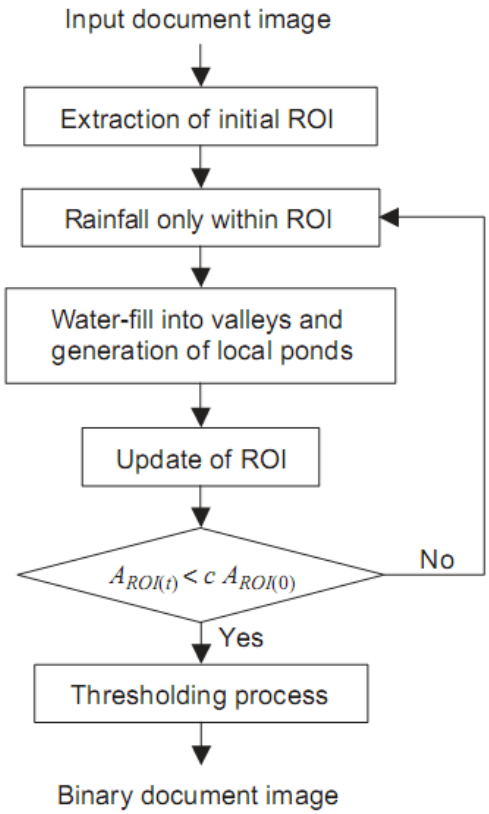


Figure 2.12 Flow Chart of the Improved Method Proposed by Oh et al. (2005)

Basu et al. (2007) continued to use this idea to deal with the problem of text line extraction from optically scanned document images. Because of the appearance of multi-skewed lines in text images, the problem becomes complex and difficult to solve completely. To solve this problem, Basu et al. (2007) proposed a text line extraction technique based on a water flow model. In this technique, hypothetical water flows, which are required to move from both left and right sides of the image frame, are stuck by the characters of text lines. Then, the boundaries of un-wetted areas on the image frame are labeled to extract text lines. Brodic and Milivojevic (2009[a] and 2009[b]) also adopted the method of Basu et al. (2007) for the problems of identifying and detecting reference text line, respectively. However, these methods limited hypothetical water flows under a few specified angles of document image frame. Thus, they failed in complex text examples in which multi-skewed lines with large skew angle exist in the text images. Brodic and Milivojevic (2010) proposed a modification of the water flow method for segmentation and text parameters extraction of sample text at almost any skew angle. The modification includes the extension of skew angles of the document image frame and the enlargement of un-wetted image regions. The procedure of document text image identification proposed by Brodic and Milivojevic (2010) is shown in Figure 2.13.

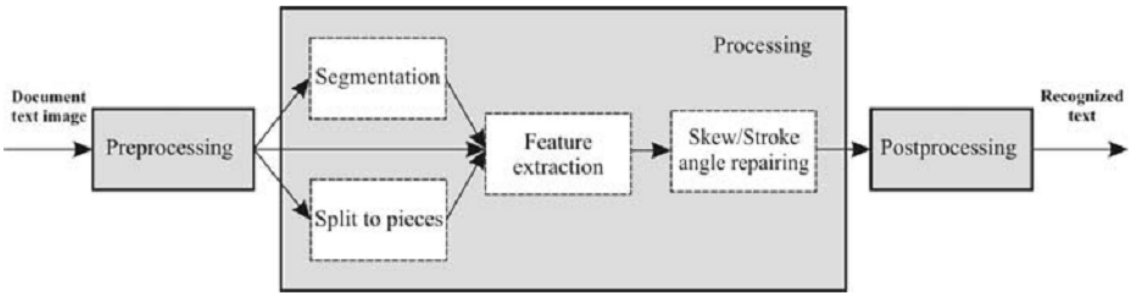


Figure 2.13 Procedure of Text Image Identification (Brodic and Milivojevic, 2010)

In summary, the works in this section only focus on combining the property of water flow always moving down to lower regions and filling up valleys into their model or technique for pattern recognition problems. For the document image processing problem in which the objective is to extract characters from backgrounds, the valleys with the amount of filled water are extracted. Otherwise for the problem of text line extraction from optically scanned document images, un-wetted areas on the image frame are labeled to be extracted. The models or techniques are simple and have not been established as algorithmic methods.

2.3.2 Intelligent Water Drops Algorithm

The idea of constructing an optimization algorithm based on certain factors or processes related to water flow was developed by Shah-Hosseini (2007). In this paper, the author proposed an intelligent-water-drops (IWD) algorithm based on the dynamics of river systems and the actions/reactions that happen between water drops and the environmental changes in the flowing river. In this algorithm, artificial water drops are designed with two properties of natural water drops, i.e., moving velocity and an amount of carrying soil. The values of both properties may change when the water drops travel in the river since an amount of removed soil from the river increases their velocity and volume. To travel from the source to the destination, the water drops choose the paths with minimal soil. A flow chart of the IWD algorithm is shown in Figure 2.14.

The IWD algorithm has been applied to solve optimization problems, such as the traveling salesman problem (Shah-Hosseini, 2007), the multiple-knapsack problem (Shah-Hosseini, 2008), and the *n*-queen puzzle (Shah-Hosseini, 2009). However, the properties

of water flow and meteorological phenomena related to water drops have not been explored fully in these papers. Also, this algorithm depends on the large number of parameters defined by users, which may affect the performance of the algorithm. Furthermore, the computational results obtained by the IWD algorithm for its applications have not been good when compared with other algorithms.

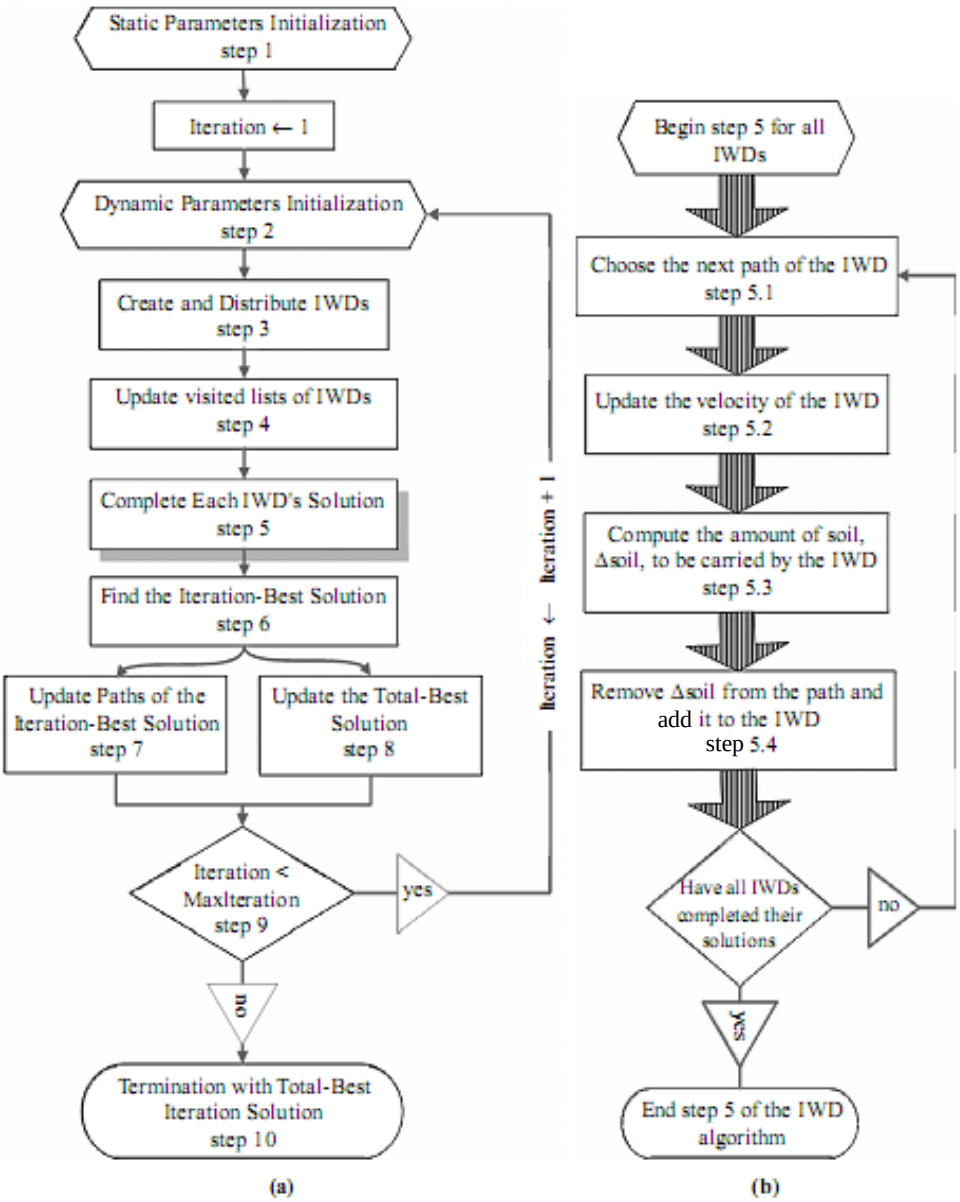


Figure 2.14 Flow Chart of the IWD Algorithm (Shah-Hosseini, 2007)

2.3.3 Water Flow-Like Algorithm

An optimization algorithm based on the behaviors of water flow was introduced by Yang and Wang (2007) for solving the bin packing problem, which is one of the well-known discrete optimization problems. This algorithm is a multiple-agent-based optimization method with an improvement idea of using a population of agents with size that is not fixed. This algorithm may be described as follows: solution agents are modeled as water flows which move on the terrain. Driven by the gravity and governed by the energy conservation law, water flows will automatically move to lower altitudes on the terrain. When moving from higher altitudes to lower ones on a rugged terrain, the water flow will split into multiple sub-flows. Fluid momentum helps water flows adjust their compositions and directions when they move through the rough terrains. When a number of water flows move to the same position, they will be merged into a single water flow. In addition, water flows may move upward to higher altitudes if their kinetic energy is larger than the potential energy required. Furthermore, to explore search space, some water flows may periodically evaporate and return to the terrain by precipitation.

In general, this is an evolutionary algorithm involving four operations: splitting and moving, merging, evaporation, and precipitation. These operations are established and controlled by some fundamental laws of physics, such as the law of energy conservation, and the law of momentum conservation. Physics quantities such as mass, velocity, fluid momentum, energy, and gravitational acceleration are also used as basic parameters to construct this algorithm. A flow chart of the algorithm proposed by Yang and Wang (2007) is shown in Figure 2.15.

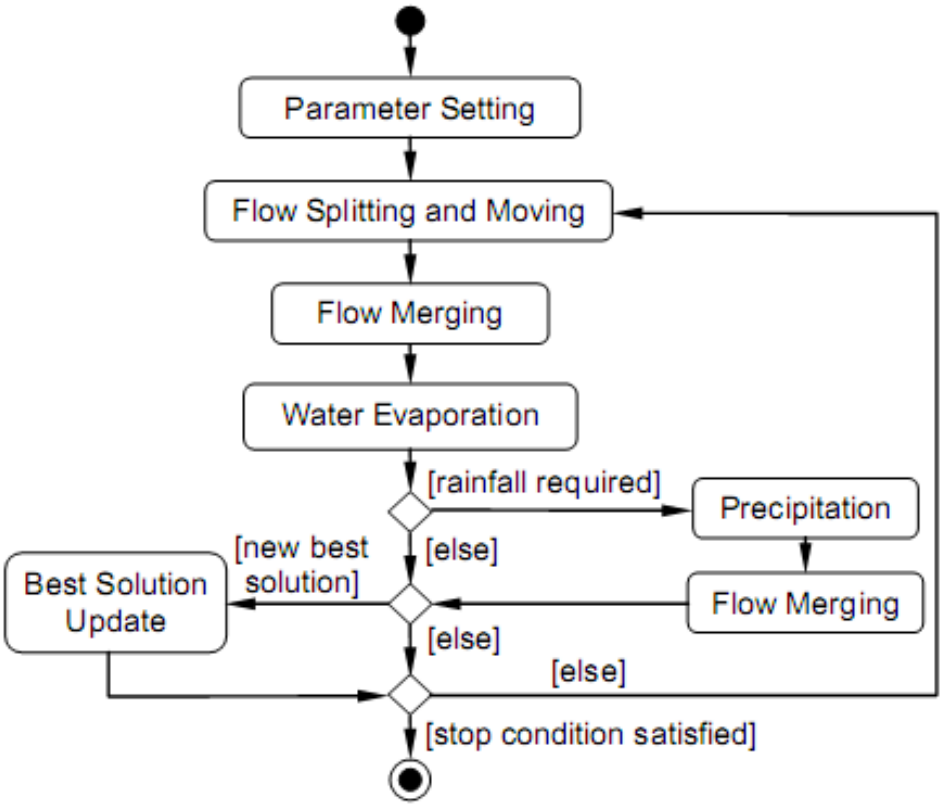


Figure 2.15 Flow Chart of the Water Flow-Like Algorithm (Yang and Wang, 2007)

The experimental results and comparisons showed that this algorithm can perform well for solving the bin packing problem. However, Yang and Wang (2007) only used two test instances to evaluate and compare their algorithm with other algorithms, and may not be sufficient to conclude that their algorithm is efficient for the bin packing problem. Also, because of the dependence on the large number of controlled parameters defined by users, this algorithm may spend a considerable amount of computational time for evaluating and processing the operators.

Wu et al. (2010) adopted the logic of the algorithm proposed by Yang and Wang (2007) to design a heuristic algorithm for solving the cell formation problem. This is one

of the important problems in cellular manufacturing when group technology is applied. The proposed algorithm consists of two stages. In the first stage, the algorithm generates initial feasible solutions in order to determine optimal cell size. In the second stage, the optimal cell size obtained is used as a lower bound to search for optimal solutions. Since the cell size may be a good lower bound for the solution process, the computational time needed by the algorithm is significantly reduced, especially for large-sized instances.

The experimental results and comparisons for a set of 37 test instances from the literature showed that the proposed algorithm of Wu et al. (2010) has performed better than other algorithms, especially for large-sized problems. However, the large number of controlled parameters still present in this algorithm may limit the performance of the algorithm.

From the success obtained for solving optimization problems, such as the traveling salesman problem, the bin packing problem, the multiple-knapsack problem, the *n*-queen puzzle and the cell formulation problem, we can see that the algorithms inspired by the behavior of water flows in nature indicate a promising optimization approach.

2.4 Conclusions and Possible Nature-Inspired Algorithm

A summary of the nature-inspired algorithms described in this chapter is shown in Table 2.1. In addition to the introduction of the basic ideas of constructing the algorithms, we also provide the most important applications for which each algorithm has worked best. This helps researchers have a condensed but ample overall view of the real-world application capability of the nature-inspired algorithms.

Also, we present the timeline of all the nature-inspired algorithms presented in this chapter (see Figure 2.16). From this figure, we can see that the optimization approaches have recently developed significantly in terms of quantity as well as quality. Many problem-solving models from nature are inspired by the design of optimization algorithms. Successful applications of the algorithms for continuous and discrete optimization problems in engineering have demonstrated the efficiency of these algorithms. Although water flow inspired algorithms are developed recently, the algorithms have showed their efficiency and effectiveness for solving some well-known optimization problems, such as traveling salesman problem and bin packing problem. However, the potential of this algorithm has not been fully exploited since many characteristics as well as behaviors of water flow have not been considered. Hence, constructing an optimization algorithm based on water flow models is still a promising research area.

A possible nature-inspired algorithm will be to integrate many additional features of water flow in nature. Natural phenomena, such as the hydrological cycle and erosion process, which have not been explored by other algorithms, will also be simulated in such a proposed algorithm. Two major phases of the algorithm, the solution exploration and exploitation phases which are inspired by the hydrological cycle and the erosion process respectively, help to achieve a balance between solution diversification and intensification capabilities to search for optimal solutions in reasonable computation time.

Moreover, this algorithm has only a small number of parameters defined by users, which may limit the performance degradation of the algorithm due to the tuning of the parameters. In addition, this algorithm can be applied to solve different optimization

Table 2.1 Summary of Applications of the Nature-Inspired Algorithms

Groups	Sub-groups	Algorithms	Ideas	Problems
Biologically inspired algorithms	Evolutionary algorithms	Genetic algorithm	The evolutionism of Darwin	Permutation flow shop scheduling (Reeves, 1995)
		Memetic algorithm	Genetic algorithm + local search	Traveling salesman problem (Moscato & Norman, 1992)
		Shuffled frog-leaping algorithm	Memetic algorithm + the social behavior of frogs	Water distribution network (Eusuff & Lansey, 2003)
	Stigmergic optimization algorithms	Termite algorithm	Process of nest reconstruction of termites	Mobile wireless ad-hoc networks (Roth & Wicker, 2006)
		Ant colony optimization	The foraging procedure of ants	Traveling salesman problem (Dorigo & Gambardella, 1997)
		Bee colony optimization	The behavior of honey bees in foraging/mating	Telecommunication network (Farooq, 2006)
	Swarm-based optimization algorithms	Particle swarm optimization	The social behavior of a school of fish or a flock of bird	Flow shop scheduling (Liao et al., 2007)
		Firefly algorithm	The social behavior of fireflies	Continuous constrained optimization (Lukasik & Zak, 2009)
		Bat algorithm	The echolocation behavior of bats	Continuous constrained optimization (Yang, 2010)
Botanically inspired algorithms		An invasive weed optimization	The ecological process of weed colonization and distribution	Electromagnetic designing problem (Karimkashi & Kishk, 2010)
		A botany-grafting inspired algorithm	The idea of grafting in botany	Spherical conformal array (Li et al., 2010)
Water flow inspired techniques		Intelligent water drops algorithm	The dynamics and reactions of river systems	Traveling salesman problem (Shah-Hosseini, 2007)
		Water flow-like algorithm	The energy conservation law	Cell formation problem (Wu et al., 2010)

problems without much change in the structure of the algorithm. These are features found lacking in most existing water flow inspired algorithm.

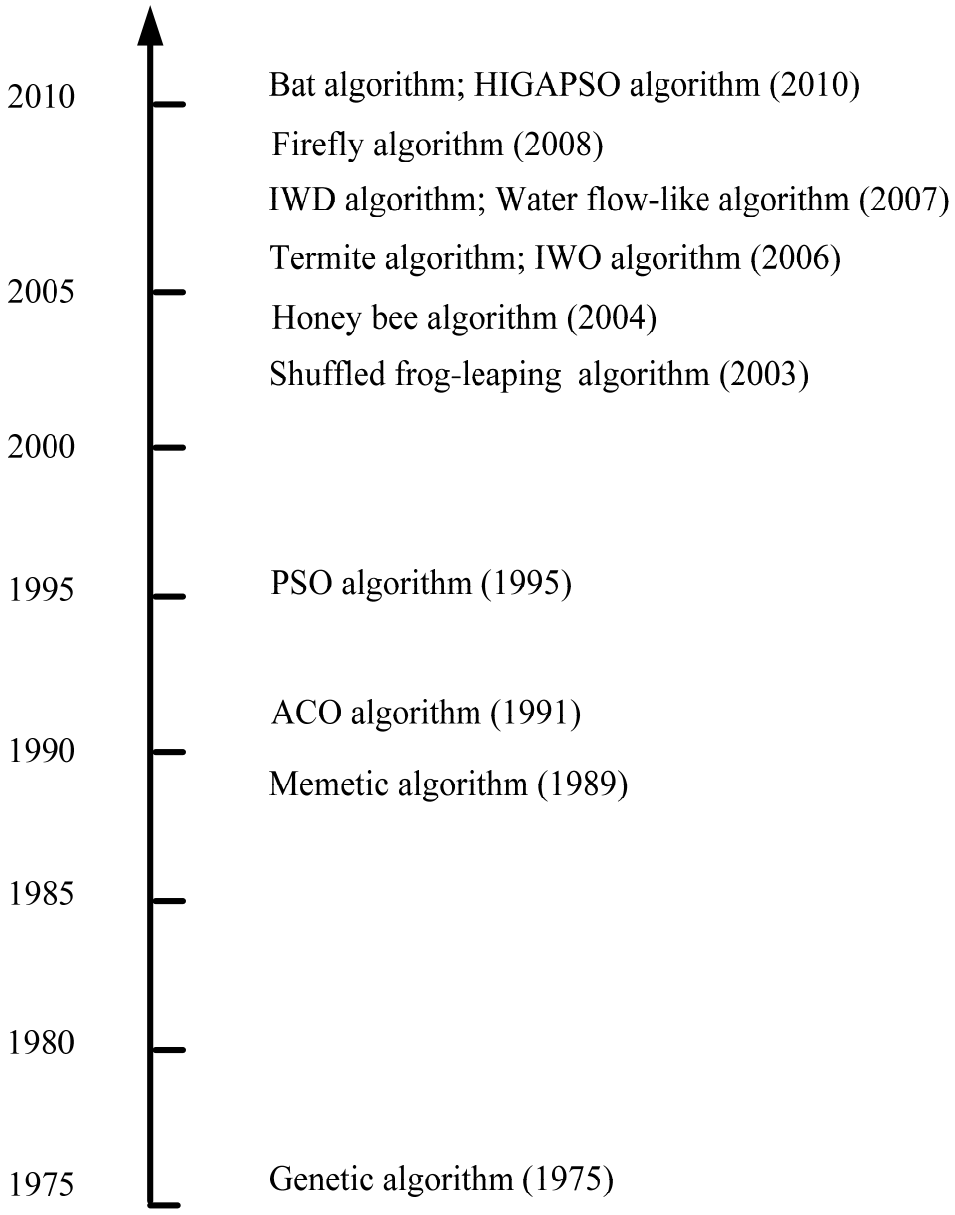


Figure 2.16 Timeline of Nature-Inspired Algorithms

CHAPTER 3

A GENERAL WATER FLOW ALGORITHM

In this chapter, we introduce a novel nature-inspired algorithm for solving combinatorial optimization problems. This algorithm simulates the hydrological cycle in meteorology and the erosion phenomenon of water flow in nature, which represent the solution exploration and exploitation capabilities of the algorithm, respectively. Being inspired by the behaviors and characteristics of water flow in meteorology and nature, this algorithm is named water flow algorithm (WFA). A detailed description of the natural phenomena imitated to construct this algorithm is provided in the chapter. In Section 3.1, we present the hydrological cycle in meteorology together with the components that constitute the hydrological cycle, such as evaporation, condensation, transportation, precipitation, transpiration, groundwater and run-off. The erosion phenomenon of water flow in nature is described in Section 3.2. Finally, a detailed description of the general WFA for solving combinatorial optimization problems is presented in Section 3.3.

3.1 Hydrological Cycle in Meteorology

Water is the source of all life on Earth. Hence, water plays an essential role in the living of creatures as well as human beings. In nature, water exists in three states: solid, liquid and gas. These states are affected by changes in the temperature of the environment. When temperature increases, water can change from the solid state to the liquid state due to melting, or from the liquid state to the gaseous state due to evaporation. Otherwise, when temperature decreases, water can change from the liquid state to the solid state due to freezing, or from the gaseous state to the liquid state due to condensation. When temperature drops below the freezing point, water molecules can change directly from the gaseous state to the solid state due to sublimation. Such changing of the states of water is a fundamental concept in the explanation of many meteorological phenomena.

One of the well-known meteorological phenomena is the hydrological cycle. The hydrological cycle reflects the circulation and conservation of water on Earth. Though water in ocean, river, cloud and rain is often in a state of change, the total amount of water on Earth is not changed. Due to the conservation of water, the circulation of water is established. The hydrological cycle consists of the following main components.

(1). Evaporation: a process that transfers water from the liquid state in ocean to the gaseous state in atmosphere.

(2). Condensation: a process that transfers water in atmosphere from the gaseous state to the liquid state.

(3). Transportation: a movement of water vapor from ocean to land through atmosphere due to wind.

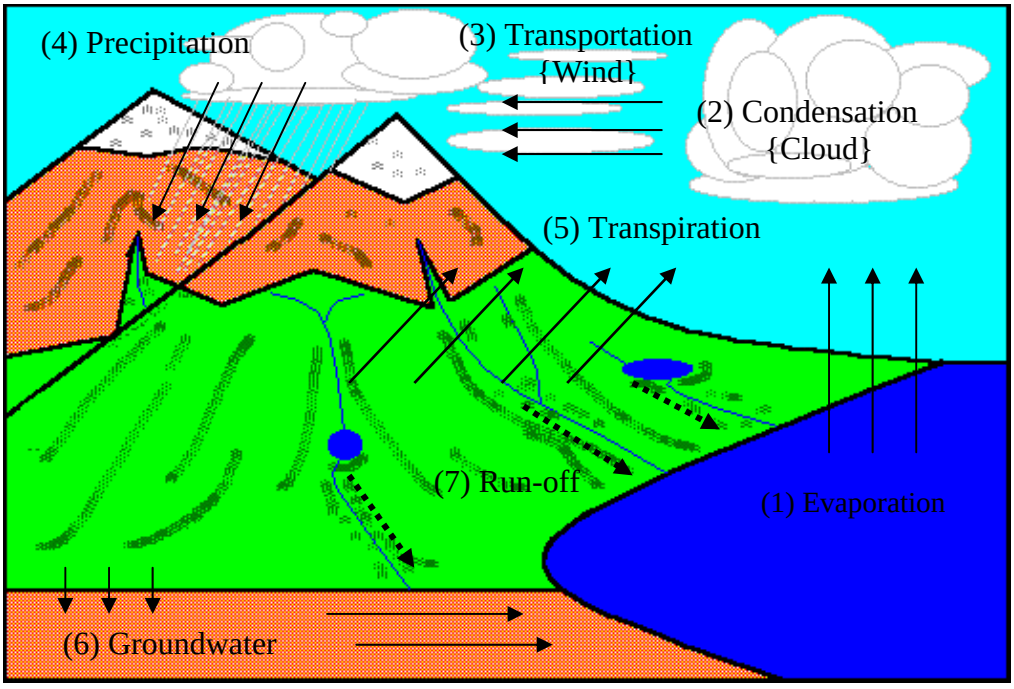
(4). Precipitation: a process that transports water from atmosphere to the Earth's surface, also called rain.

(5). Transpiration: a process that transfers water from ground to atmosphere by the evaporation of plants and vegetation.

(6). Groundwater: the amount of water that infiltrates through the Earth's surface to return to the ocean.

(7). Run-off: the rest of the amount of water on the Earth's surface in rivers, lakes, ponds or streams, after the transpiration and infiltration processes occur.

Figure 3.1 illustrates these seven basic components of the hydrological cycle. The exploration phase of WFA proposed in this thesis is inspired by the hydrological cycle. The phase is designed based on the basic components.



(Source: http://ww2010.atmos.uiuc.edu/(Gh)/guides/mtr/hyd/smry.rxml)

Figure 3.1 Basic Components of the Hydrological Cycle

A detailed description of the meteorological phenomenon is provided in Goudie (1993). Here, we briefly present the relationship among the main components mentioned in the hydrological cycle.

Firstly, sea water from the surface of the oceans is evaporated to become water vapor in the atmosphere. When the water vapor is lifted and encounters cool air, it is condensed to form clouds with moisture. Then, the moisture in clouds is transported around by wind until it returns to the Earth's surface under drops of water through precipitation. When the drops of water fall down to the ground, they are subjected to two possible processes. In the first process, some of the drops of water may be evaporated back into the atmosphere through transpiration. In the second process, some of the drops of water may penetrate through the Earth's surface to be groundwater. The drops of water returning to the atmosphere continue to join the clouds and move around, while the groundwater moves back to the oceans.

The rest of the water on the Earth's surface is called runoff. Governed by the gravity force, the runoffs move from higher altitudes to lower ones on the ground. Lakes, rivers or streams are formed from the runoffs if they are held in valleys or depressions. When the amount of precipitation increases dramatically, water in the lakes, rivers or streams may flow over their barriers and move back to the oceans, where the hydrological cycle begins again. This overflowing phenomenon is known as erosion process of water flow in nature, which will be described in the next section.

3.2 Erosion Process of Water Flow in Nature

In nature, there are many phenomena affecting and changing the Earth's surface. Erosion process is one of such natural phenomena. In essence, erosion is a natural process of weathering to the Earth's surface, which detaches solids from the surface and deposits them elsewhere (Holy 1982). This process is caused by movements of wind and water in natural environment. In addition, the gravity force of Earth and the operations of living creatures are the causes of the erosion process. However, since water covers over 70% of the Earth's surface area (Michael, 2006), we can see that it is the major cause of the erosion process. Figure 3.2 shows an illustration of the erosion by water flow.

Figure 3.2

Erosion from Water Flow

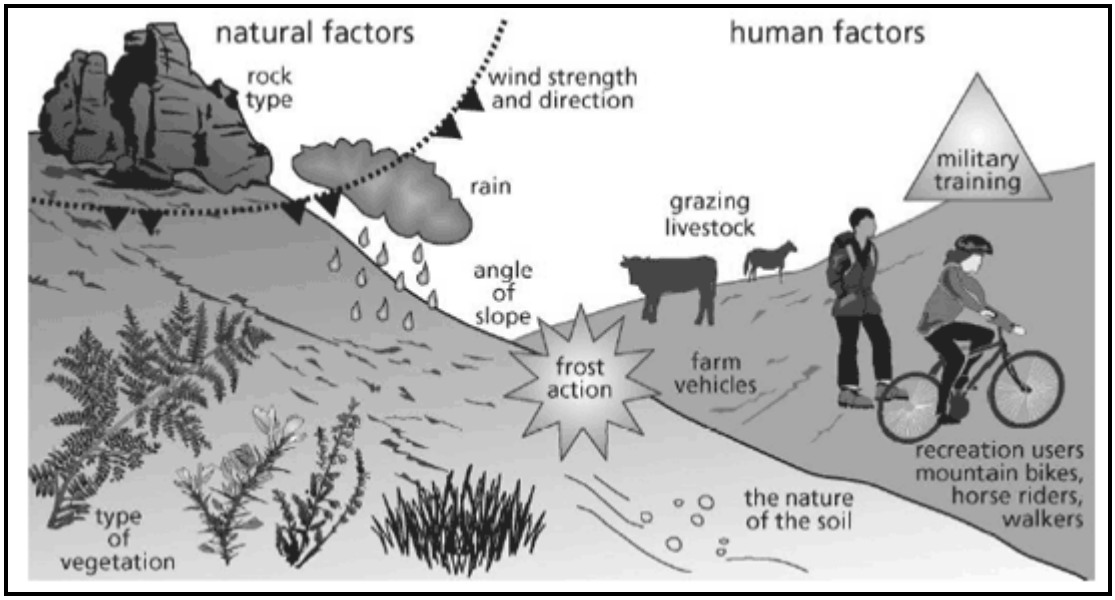
(Source: http://www.salomart.com/images/erosion-2.jpg)

Normally, the consequence of erosion process caused by water is not serious to human beings since it occurs slowly and gently in rivers or streams. However, when an amount of water of the rivers or streams increases dramatically due to heavy precipitation or tide, the erosion process may cause a flood. The flood can sweep away everything on the path through which it passes.

The most important property of the erosion process caused by water flow is erosion rate, or known as erosion capability, which is also a factor to recognize and control the flood. The erosion rate is determined by an amount of water in watercourse, the velocity of water streaming, as well as the state of land. In nature, an amount of water in watercourse is affected by the amount of precipitation, intensity of precipitation and flood-tide; while the velocity of water streaming is mainly affected by the slope of landscape (or gravity force) and the amount of water in the watercourse. Here, the state of land merely means the hardness of land. On the other hand, the erosion rate depends on three major groups of factors, i.e., climatic factors, geologic factors and biological factors (Holy, 1982). The climatic factors include the amount and intensity of precipitation as mentioned above. In addition, average temperature, wind speed, as well as storm frequency may be considered as climatic factors affecting the erosion rate. The geologic factors consist of the types of sediment or rock, their porosity and permeability, the slope of landscape, as well as the shape of rocks. The biological factors are the ground cover area of vegetation, the type of organisms living in the region, and the purpose of using land.

A different classification of factors that involves the impact of erosion capability is provided in Figure 3.3. Here, these factors are divided into two main groups, i.e., natural factors and human factors. Despite the classifications, it can be seen that there are many

factors affecting the erosion capability. Hence, it is not easy to control the erosion rate/capability and flooding in general.



(Source: http://www.dartmoor-npa.gov.uk/learningabout/lab-printableresources/lab-factsheetshome/lab-erosion)

Figure 3.3 Factors Affecting Erosion Capability

There are also many studies on the relationship between erosion process and factors affecting this process. Such studies show that when the amount and intensity of precipitation increase, erosion rate will increase. On the other hand, erosion rate is also expected to adjust in response to climate changes due to many reasons. The most direct reason is the change in the erosive power of precipitation. Other direct reasons include:

(1). Because of shifting precipitation regime and evaporation/transpiration rate, land moisture is also changed. It affects infiltration and runoff ratios.

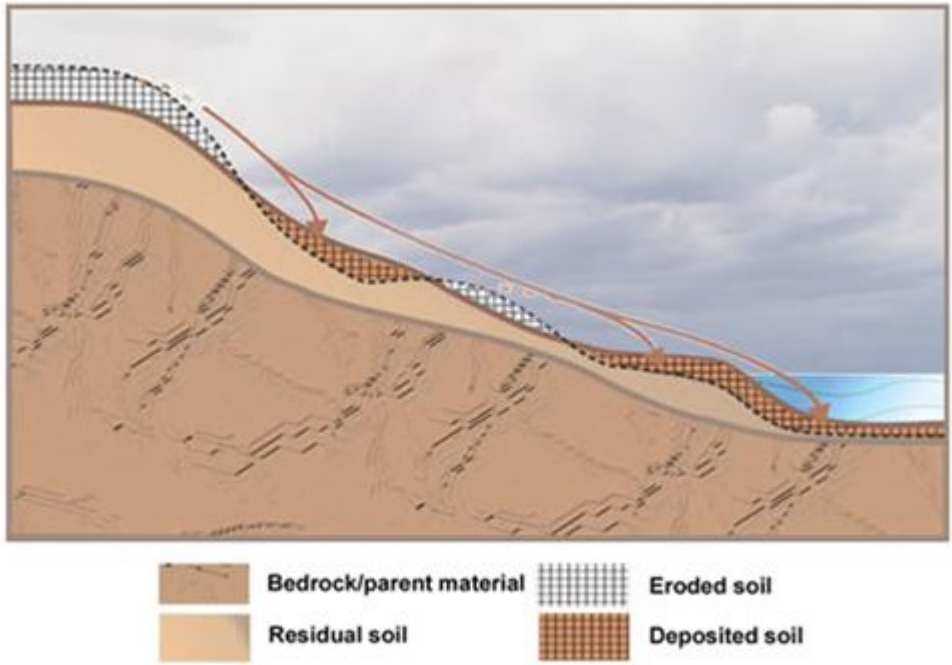
(2). Because of reducing concentration of land, land erodibility increases. Also, it leads to increase in the amount of runoff due to appearance of rifts on the surface.

(3). Due to increasing temperatures, winter precipitation is shifted from non-erosive snow to erosive rainfall.

(4). Due to melting of permafrost, a non-erodible land state may be shifted to be an erodible land state.

All these reasons help to understand clearly about the close relationships of erosion rate and other factors. We may then propose methods to control the erosion rate, or at least to limit the loss caused by increasing the erosion rate.

In summary, erosion process is the displacement of solids usually caused by the water currents, as well as the down-slope movement of the solids in response to gravity. The erosion process is able to affect the Earth's surface and change the direction of water current. In addition, erosion process is able to smooth out obstacles and keep a state of smoothed terrain for a long time period. Figure 3.4 shows an illustration of the terrain smoothed by the erosion of water flow. Erosion rate, an important parameter of the erosion process, is affected by many factors. The factors consist of the amount and the intensity of precipitation, the texture of soil, the gradient of slope, etc. However, only two major factors have been considered to design the capability of erosion process in the WFA. These factors are the amount of precipitation as well as the texture of soil, including the hardness of soil and the geographical shape of the surface. How to simulate the hydrological cycle as well as to integrate the erosion process of water flow in constructing an optimization algorithm is presented in the next section.



(Source: http://newscenter.lbl.gov/feature-stories/2007/04/22/damaged-land-buried-carbon/)

Figure 3.4 Illustration for Smoothed Terrain by Erosion Process

3.3 General Water Flow Algorithm

In this section, we present a general water flow algorithm for combinatorial optimization problems. The framework of the WFA can be customized and used to solve various types of optimization problems.

The WFA mimics the hydrological cycle in meteorology and the erosion phenomenon in nature, representing a balance between solution exploration and exploitation capabilities

in an optimization algorithm, respectively. The proposed algorithm is based on the simulation of spreading raindrops into many places on the ground, the property of water flow always moving from higher positions to lower positions, and the erosion capability of water flow on the ground. The following descriptions further illustrate some of the relationships between the terms used in the WFA and the concepts often used in other well-known meta-heuristics, such as tabu search and genetic algorithm:

(1). The hydrological cycle reflects the circulation and conservation of water on the Earth. It consists of many stages, i.e., evaporation, condensation, transportation, precipitation, transpiration, groundwater and run-off (Goudie 1993). In the WFA, we use the circulation of water to regenerate a population of new positions for drops of water (*DOW*s) after each cloud is generated. It is similar to the diversification procedure of restarting an initial solution in tabu search if the current solution is not improved after a maximum number of iterations.

(2). The precipitation is the transfer of water from the atmosphere to land, usually called rain (Goudie 1993). When there is rain, an amount of raindrops is generated and it drops to the ground. In the WFA, the precipitation involves the number of *DOW*s generated. It can be considered as generating a population of solutions in a genetic algorithm.

(3). The concept of erosion is the detachment of soil particles caused by the impact of water flow or raindrops (Holy 1982). The role of erosion in the WFA is to help the search to intensify the promising local optimal positions and overcome the obstacles to find better positions. It is similar to intensification in tabu search.

(4). The rate or intensity of erosion in nature depends on many factors, such as the amount of precipitation and the soil type, etc. (Holy 1982). In the WFA, this erosion capability is represented by the maximum number of iterations before erosion process stops eroding at the local optimal position being considered. It is similar to the concept of the maximum number of iterations without improvement used in tabu search.

We now describe the general procedure of the WFA. Firstly, a cloud representing an iteration randomly generates a set of *DOW*s onto some positions on the ground, which represent solutions of the optimization problem. Next, due to the gravity force of the Earth represented by a heuristic algorithm, the *DOW*s automatically move to local optimal positions. They are held at these positions until the erosion condition is satisfied before performing the erosion process. Then, depending on the amount of precipitation, the falling force of precipitation and soil hardness at the local optimal positions, the erosion process helps the *DOW*s overcome the local optimal positions to find better or global ones. Almost all the ideas of constructing this procedure are novel, except for the idea of water flow always moving to lower positions, which is similar to the previous works.

A flow chart of the general WFA for solving optimization problems is shown in Figure 3.5, while the pseudocode of the WFA is presented in Figure 3.6. The detailed descriptions of the erosion condition, erosion capability, erosion process and other operations of the general WFA for optimization problems are provided in the next subsections.

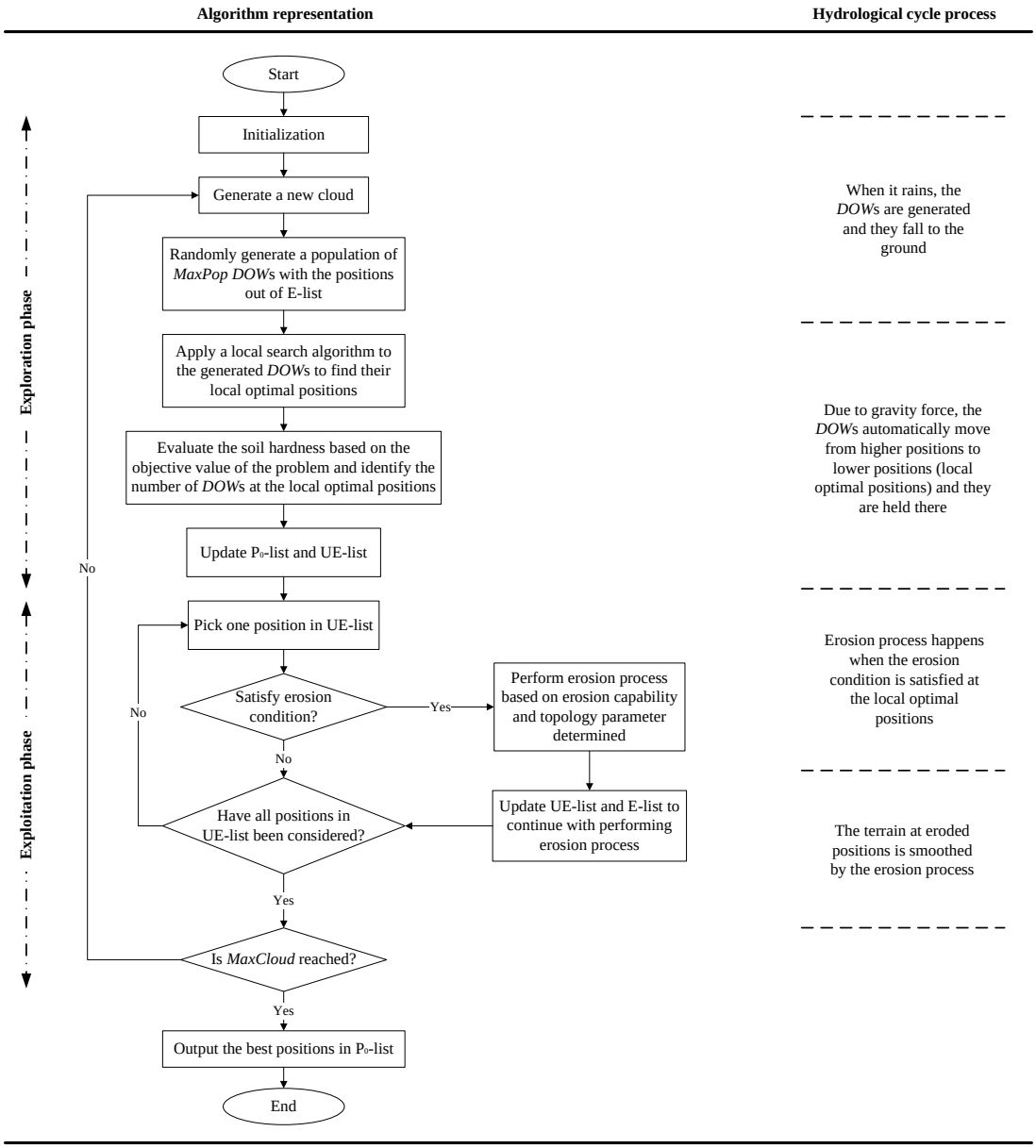


Figure 3.5 Flow Chart of the General WFA

3.3.1 Encoding Scheme

For any optimization algorithm, encoding scheme, also known as solution representation, is the first important step in the implementation of the algorithm. In the WFA, we have

Step 1:

Initialize the controlled parameters, i.e., the maximum number of clouds generated (*MaxCloud*), the maximum number of *DOW*s that each cloud is allowed to generate (*MaxPop*), the minimum number of *DOW*s in which the erosion process starts to perform (*MinEro*), the maximum iterations that the erosion process will move to next erosion direction if the position considering is not improved (*MaxUIE*);

Step 2:

For *i* = 1 to *MaxCloud*

Randomly generate a population of *MaxPop* positions for *DOW*s;

Determine the number of *DOW*s at each individual position;

Find the local optimal position for each individual position;

Update the best positions in P_0-list;

Update the local optimal positions found and the number of *DOW*s into UE-list;

For each position in UE-list

If (satisfying the erosion condition) **Then**

Perform erosion process;

Update UE-list, P_0-list, and E-list;

End if

Next each

Next *i*

Step 3:

Return the best positions found in P_0-list.

Figure 3.6 Pseudocode of the General WFA

used *DOW*s to represent solutions of an optimization problem. Then, all the operators of the WFA are performed on the *DOW*s. Depending on the structure of a certain problem, *DOW*s are appropriately encoded to solve the problem efficiently.

In nature, the positional information of a *DOW* is shown by its longitude, latitude and altitude on the ground. In an optimization problem, a *DOW* is associated with a feasible solution of the problem. We consider the feasible solution and its objective value as providing the longitude, latitude and altitude information for the position of the *DOW* on the ground. For example, when applied to solving flow shop scheduling problems, a job permutation will provide the longitude and latitude information for a *DOW*; while the objective value, i.e., the completion time of jobs obtained by the corresponding job permutation, will provide the altitude information for the *DOW*.

3.3.2 Memory Lists

In the WFA, three memory lists are used to support the search for global optimal positions of *DOW*s. Firstly, the best position list, called the P_0-list, stores the best optimal positions found so far. This list is almost used in all optimization algorithms to output the best solutions obtained by the algorithms. Secondly, the un-eroded position list, called the UE-list, is used to store the local optimal positions which have not been eroded because of not satisfying the erosion condition. Its purpose is to record the potential positions for the subsequent erosion process of the WFA. The reasons for using local optimal positions as potential positions/regions will be described and discussed later. Thirdly, the eroded position list, called the E-list, is used to store eroded local optimal positions. The E-list is inspired by terrain smoothing of erosion process. It plays an important role of preventing

*DOW*s from being regenerated to the eroded positions in the subsequent precipitations. It would help to reduce the computation time needed by the algorithm.

3.3.3 Exploration Phase

The exploration phase of the WFA is inspired by the hydrological cycle in meteorology presented in Section 3.1. The number of basic components of the hydrological cycle used in the WFA depends on the optimization problem to be solved. For example, when solving single-objective optimization problems, only transportation, precipitation and run-off of the hydrological cycle are utilized in the exploration phase. However, when solving multi-objective optimization problems, the exploration phase may include evaporation/ transpiration, condensation and groundwater. Despite the number of basic components that are completely used, the general procedure does not undergo much change.

The exploration phase can be described as follows: at each cloud, we generate randomly a set of positions for *DOW*s with size *MaxPop*. This is to simulate the spreading of raindrops into many places on the ground. Based on the problem to be solved, we can use an efficient constructive method to generate a set of seed positions for *DOW*s at the first cloud, which may improve the performance of the WFA. After generating a population of positions for the *DOW*s, a steepest descent hill sliding algorithm is used to search for local optimal positions from these initial positions of the *DOW*s. In particular, the hill sliding algorithm searches for the best improved position within the initial position's neighbors in terms of objective value. Then, this process is iteratively performed in the same manner until no other improved position is found. Depending on the problem, an appropriate neighborhood structure is proposed. The idea of determining

local optimal positions of *DOW*s is inspired by the property of water flow always moving to lower positions.

In general, the exploration phase of the WFA results in a set of local optimal positions of *DOW*s. The local optimal positions and the number of *DOW*s at these positions are updated in the UE-list to be considered for performing the erosion process in the next exploitation phase.

3.3.4 Exploitation Phase

Exploitation phase of the WFA is inspired by the erosion phenomenon of water flow in nature presented in Section 3.2. The properties of the erosion phenomenon are simulated in this phase of the WFA to guide *DOW*s to overcome the local optimal positions and search for better or optimal positions. In the next subsection, we present the conditions to perform erosion process for the local optimal positions in the UE-list. Also, the calculations of erosion capability for the *DOW*s at the positions satisfying the conditions, as well as the basic operational mechanism of the erosion process are described. Depending on the nature of the problem, the corresponding operators of the erosion process will be implemented to solve the problem efficiently.

3.3.4.1 Erosion Condition and Capability

Although many factors have effects on the erosion process of water flow in nature as described in Section 3.2, only two major factors affect the erosion process significantly. They are the hardness of soil and the amount of precipitation. In nature, the hardness of soil varies according to the land area considered. However, when solving an optimization

problem representing the topography of the Earth, the landscape of the problem does not change over time and space. Thus, we assume that the soil hardness value at every position in the problem is the same. Thus, we are only concerned about the amount of precipitation for determining the erosion condition to perform the erosion process in the WFA. If the amount of precipitation at some local optimal position increases up to *MinEro*, the erosion process would happen at the local optimal position.

Also, many factors affect the erosion rate/capability of water flow as presented in Section 3.2. However, for simplicity, we only consider the capability of erosion process based on two main factors: the amount of precipitation and its falling force. In optimization problems, the amount of precipitation is represented by the number of *DOW*s at the eroding local optimal position; while its falling force is represented by the objective value of the corresponding problem at the position. We will next explain why the objective value at the corresponding position is considered as the falling force of precipitation.

In the WFA, the altitude information of a *DOW* is determined by the corresponding objective value of a solution in an optimization problem. For the problem of minimization, the *DOW* with the smaller altitude leads to the better quality solution. Moreover, the falling force of raindrops depends on the distance between the cloud and the position of the raindrops in which they fall on the ground. Hence, if we assume that the clouds move at the same altitude, the impact of falling force at the lower positions is larger than that at the higher positions. In other words, the erosion capability at the lower positions is larger than that at the higher positions. This is the reason why the falling force of precipitation is represented by the objective value at the corresponding position. Figure 3.7 illustrates the

erosion capability at two positions with two different altitudes. In Figure 3.7, the erosion capability at point B is larger than that at point A.

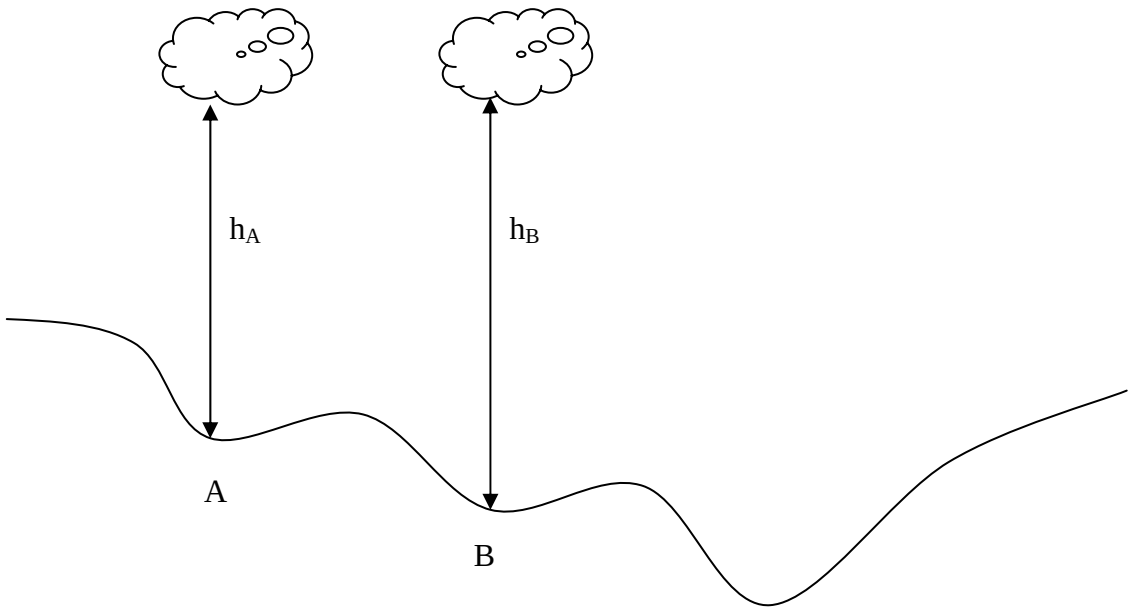


Figure 3.7 Illustration for Effect of Altitude on Erosion Capability

With the assumption that clouds are at the same altitude, the falling force of precipitation to lower local optimal positions will cause erosion more easily than that of higher ones. Then, we can infer that the erosion capability becomes stronger for local optimal positions with the larger amount of precipitation and the lower objective values. It may create a flexible operation scheme for the erosion capability of *DOW*s in the WFA. Also, it helps the erosion process focus on exploiting promising regions strongly while ignoring regions with poor performance. In particular, the relationship between these two factors and the control parameter of erosion capability, *MaxUIE*, is expressed as follows:

$$MaxUIE = \varphi_1 Q(\pi^*) + \varphi_2^{LB/z}, \tag{3.1}$$

where φ_1 and φ_2 are parameters representing the effect of precipitation and its falling force, respectively. Also, $Q(\pi^*)$ is the number of *DOW*s at the local optimal position π^*, *LB* is a known lower bound, and *z* is the objective value at the eroding local optimal position. Here, *LB* can be any lower bound obtained from the literature, or by solving a relaxation of the problem in terms of the corresponding objective function.

In addition, we also propose a simple determination of erosion capability for solving optimization problems with less complex structure. In this process, the erosion capability does not depend on the amount of precipitation and its falling force. Here, we use a constant value pre-specified for the erosion capability. When the amount of precipitation at some local optimal position increases up to *MinEro*, the erosion process with the pre-specified erosion capability would happen at the local optimal position. It is helpful for optimization problems with less complex structure since the computation time may decrease without affecting the solution quality obtained.

3.3.4.2 Erosion Process

Erosion process will be performed when the erosion condition at some local optimal position is satisfied. Depending on the structure of the problem, the erosion capability at the local optimal position may either be constant or determined from equation (3.1). The strategy of erosion process is based on a topological parameter representing the geographical surface, and whether an erosion direction is blocked.

The topological parameter Δd is defined as the difference between the objective value of local optimal position and that of its neighboring position. We first choose the smallest

Δd as the erosion direction. If the erosion process for that direction does not improve after *MaxUIE* iterations, we say that the direction is blocked. It means that water flow cannot move in that direction and searching in that direction stops. This is followed by backtracking, in which we restart the search from the local optimal position using the direction with the next smallest Δd. If all directions for considering the local optimal position are blocked, we call that position fully blocked and move it into the E-list, i.e., we do not consider that position in the subsequent clouds/iterations. Otherwise, if there is a direction with improvement when compared with the current local optimal position, we will choose that direction to erode permanently for the local optimal position. Here, the improvement means that the erosion process finds a new local optimal position, whose objective value is smaller than that of the current local optimal position. This new local optimal position is updated in the UE-list to continue with performing the erosion process. The erosion process simulated in the WFA is thus close to the natural behavior of water flow in the erosion phenomenon. In nature, if no unexpected change occurs, water flow always moves by following a fixed stream. The entire erosion process can be presented in pseudocode in Figure 3.8.

To perform the erosion process efficiently, we have used short-term memory in the process, called track-list. Since the nature of water flow is to never move upstream, the track-list is used to prevent the *DOW*s in the erosion process from moving back. The list only temporarily records the previously passed positions of the *DOW*s that are determined in the corresponding erosion direction. Thus, each erosion direction has a corresponding track-list. The track-list becomes empty immediately after the erosion direction is completely considered.

Procedure *Erosion Process*;

Begin

Do loop

Choose an un-eroded direction with the smallest Δd to erode;

Do loop

Apply steepest descent hill-sliding algorithm for the erosion direction chosen;

Until (new optimal position is found or no improvement after *MaxUIE* iterations)

If (the new optimal position is better than the eroding optimal position) **Then**

Update it into the UE-list to continue performing the erosion process;

Update the E-list;

End if

Until (all erosion directions are considered)

End.

Figure 3.8 Pseudocode for General Erosion Process of the WFA

The entire process of both exploration and exploitation terminates when the maximum number of allowed iterations (*MaxCloud*) is reached.

Finally, we present the reasons why the erosion process only happens at the local optimal positions with the large number of *DOW*s. The first reason is because the process is inspired by the erosion phenomenon of water flow in nature. This phenomenon occurs when the amount of water in watercourse increases dramatically as described in Section 3.2. The second reason is because the local optimal positions are considered as very

promising positions to search for global optimal positions. Intuitively, we can see that because of using random number generation for the initial population of *DOW*s, the probability of the *DOW*s generated onto every position is the same. Then, the center positions of "big valleys" may be found more easily than the ones of "small valleys" (see Figure 3.9). Also, it means that the number of *DOW*s at the center positions of "big valleys" may increase faster than that at "small valleys".

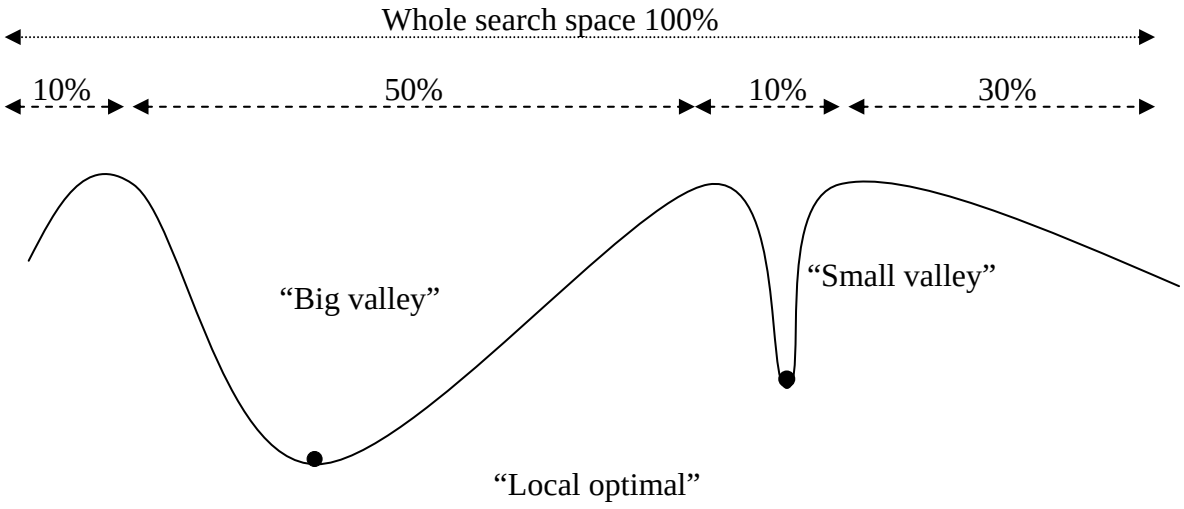


Figure 3.9 Illustration for "Big Valley" and "Small Valley"

Generally, good properties of a solution may appear many times throughout the iterations. Hence, local optimal positions found many times in the WFA can be considered as promising positions or regions to find better local optimal positions by the erosion process.

CHAPTER 4

WFA FOR PERMUTATION FLOW SHOP SCHEDULING

In this chapter, we present a water flow algorithm (WFA) for solving the permutation flow shop scheduling problem (PFSP), which is one of the well-known scheduling problems in production. The proposed algorithm has been developed based on the general framework of the WFA described in Chapter 3. The exploration phase of this algorithm is inspired by the following basic components of the hydrological cycle, i.e., transportation, precipitation and run-off; while the exploitation phase of this algorithm has used a pre-specified constant value for determining the erosion capability of erosion process.

The structure of this chapter is organized as follows. In Section 4.1, we introduce the PFSP and briefly describe a literature review of solution methods for the scheduling problem. Then, the formulation of the PFSP is presented in Section 4.2. The customization of the WFA for solving this scheduling problem is described in Section 4.3. The computational results and comparisons among the WFA and other algorithms are shown in Section 4.4. Finally, some conclusions are presented in Section 4.5.

4.1 Introduction

In production, the PFSP is one of the well-known scheduling problems in which n jobs have to be processed by m machines with the same order of jobs on machines. The most common objective of the scheduling problem is to minimize the completion time of jobs, also known as makespan ($C_{\max}$), by specifying the sequence of jobs. In addition to this objective, there are other desired measures of performance for this scheduling problem, such as minimizing total flow time of jobs or minimizing total tardiness time of jobs. Generally, the makespan criterion is often used because the minimization of makespan ensures that production gets a high throughput (Pinedo, 2002).

The PFSP with makespan minimization has been proven to be an NP-hard problem when the number of machines $m > 3$ (Rinnooy Kan, 1976), and is thus considered a challenging optimization problem. Many research works in the literature have addressed this scheduling problem, and they also proposed various optimization methods to solve it. We can classify the methods according to two approaches: exact and heuristic algorithms. Exact techniques are computationally effective for problems with small size (often less than 20 jobs). However, for problems with large size, these techniques are computationally intensive. On the other hand, as heuristic algorithms often find a good solution rapidly, it is efficient to use them to solve problems with large size.

The heuristic algorithms can be classified into three major categories: constructive algorithms, improvement algorithms and meta-heuristics. Some constructive algorithms for the PFSP include Johnson (1954), Nawaz et al. (1983) and Koulamas (1998). Among these algorithms, the algorithm by Nawaz et al. (1983) (NEH) is one of the effective

polynomial-time heuristic algorithms for the PFSP. Unlike constructive algorithms, improvement algorithms start from an initial solution, and try to improve it through some iterative procedures to obtain better solutions. Normally, the search for better solutions is based on a predetermined neighborhood structure. Taillard (1990), Ho and Chang (1991), and Suliman (2000) provide examples of improvement algorithms. Applying these algorithms for solving the PFSP has led to some success, but the quality of solutions obtained is not good for certain problems with large size. Recently, meta-heuristic algorithms, which are inspired by the behavior of natural systems, have been extensively developed and successfully applied to solve the PFSP. Some examples of the meta-heuristic algorithms are ant colony optimization (Rajendran and Ziegler, 2004), particle swarm optimization (Tasgetiren et al., 2007), and genetic algorithm (Nagano et al., 2008).

Hence, we propose a WFA for solving the PFSP. The proposed algorithm is developed based on the general WFA presented in Chapter 3. In this algorithm, the exploration phase simulates the basic components of the hydrological cycle, such as transportation, precipitation and run-off. The exploitation phase uses a pre-specified constant value for determining the erosion capability. Several well-known benchmark problem sets are used to evaluate the performance of this algorithm. The best known values of the benchmark problems in the literature are used to compare with the results obtained by the WFA. In addition, we also used these results to compare with that of other efficient algorithms, such as an adaptive learning approach combined with the NEH constructive heuristic (NEH-ALA) proposed by Agarwal et al. (2006), or a constructive GA combined with local search (CGALS) described by Nagano et al. (2008), etc.

4.2 Formulation of the PFSP

In this section, we describe and present the formulation of the PFSP. In this scheduling problem, n jobs have to be processed by m machines with the same order of jobs on machines. Let p_{ji} denote the processing time of job j (j = 1, 2,..., n) on machine i (i = 1, 2,..., m), and $C(\sigma_j, m)$ denote the completion time of job σ_j on machine m. Then, the PFSP aims to search for the best permutation of jobs processed through all machines.

Given the job permutation $\pi = (\sigma_1, \sigma_2, ..., \sigma_n)$, the completion time of the n-job m-machine problem is calculated as follows:

$$C(\sigma_1, 1) = p_{\sigma_1,1}, \tag{4.1}$$

$$C(\sigma_j, 1) = C(\sigma_{j-1}, 1) + p_{\sigma_j,1}, \quad j = 2, ..., n, \tag{4.2}$$

$$C(\sigma_1, i) = C(\sigma_1, i-1) + p_{\sigma_1,i}, \quad i = 2, ..., m, \tag{4.3}$$

$$C(\sigma_j, i) = \max\left\{C(\sigma_{j-1}, i), C(\sigma_j, i-1) + p_{\sigma_j,i}\right\}, \quad j = 2, ..., n;\ i = 2, ..., m. \tag{4.4}$$

Then, the makespan can be defined as:

$$C_{\max}(\pi) = C(\sigma_n, m). \tag{4.5}$$

Hence, the PFSP with the makespan criterion aims to search for an optimal job permutation π^* in the set of all possible job permutations Π, such that:

$$C_{\max}(\pi^*) \le C(\sigma_n, m) \quad \forall \pi \in \Pi. \tag{4.6}$$

4.3 WFA for the PFSP

In this section, we present how the proposed WFA can be applied to solve the PFSP. The basic components of the WFA described are the encoding scheme, memory lists, exploration phase, and exploitation phase.

4.3.1 Encoding Scheme

In the PFSP, a *DOW* is associated with a job permutation. We consider the job permutation and its objective value as providing the longitude, latitude and altitude information for the position of *DOW* on the ground. Given a job permutation $\pi = (\sigma_1, \sigma_2, ..., \sigma_n)$, we define:

$$\text{longitude}(\pi) = (\sigma_1, ..., \sigma_{\lfloor \frac{n}{2} \rfloor}) \tag{4.7}$$

$$\text{latitude}(\pi) = (\sigma_{\lfloor \frac{n}{2} \rfloor + 1}, ..., \sigma_n), \tag{4.8}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. The altitude of *DOW* is defined as the corresponding makespan value of the given job permutation. The objective value is calculated by equation (4.5). Figure 4.1 shows an illustrative example of a *DOW* and its positional vector components for the PFSP with 6 jobs.

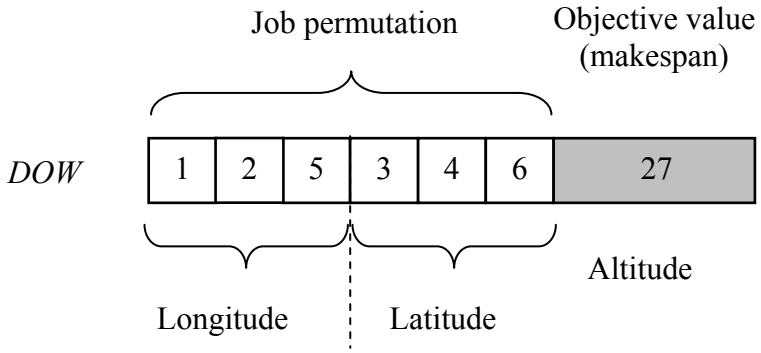


Figure 4.1 An Example of Solution Representation in the WFA for the PFSP

4.3.2 Memory Lists

In the WFA, we use three memory lists to support the search for global optimal positions. Firstly, we use one memory list to store the best positions found so far, called the P_0-list. For the PFSP, the P_0-list stores the job permutations with minimum makespan. This list is updated when a new local optimal job permutation is found. Secondly, the un-eroded position list, called the UE-list, is used to store local optimal job permutations which have not been eroded due to not satisfying erosion condition. Its purpose is to record potential job permutations for the subsequent erosion process. Thirdly, the eroded position list, called the E-list, is used to store eroded local optimal job permutations. This list aims to prevent next clouds from regenerating *DOW*s to the eroded job permutations. It would help to save the computation time needed by the algorithm. Both UE-list and E-list are updated after performing the erosion process. In addition, the UE-list is updated when a new local optimal job permutation is found in the erosion process.

4.3.3 Exploration Phase

At each cloud (or iteration), after randomly generating a population of initial job permutations for *DOW*s, the steepest descent hill sliding algorithm is used to search for local optimal job permutations from the initial job permutations. In particular, the hill sliding algorithm searches for the best improved job permutation within the initial job permutation's neighbors in terms of makespan value. Then, this process is performed iteratively in the same manner until no other improved job permutation is found. In this algorithm, a perturbation scheme based on a systematic pair-wise job exchange, a variant of 2-opt algorithm, is used to construct the neighboring job permutations. In general, the

exploration phase in the WFA for the PFSP will result in a set of local optimal job permutations. They will be updated in the UE-list to be considered for performing the erosion process in the exploitation phase.

4.3.4 Exploitation Phase

In the WFA for the PFSP, we perform the erosion process based only on the amount of precipitation. If the amount of precipitation at some local optimal job permutation in the UE-list increases up to *MinEro*, the erosion process will happen at the local optimal job permutation.

In the current WFA, we consider the capability of erosion process based on a pre-specified constant value. It means that the erosion capacity is independent of two factors, i.e., the amount of precipitation and its falling force. In particular, when the erosion process happens, the erosion capacity is determined by a constant *MaxUIE*, the maximum number of iterations for the erosion process to move to the next erosion direction if the job permutation is not improved.

In the erosion process of the WFA for the PFSP, the topological parameter Δd representing geographical surface is calculated as the difference between the makespan value of the local optimum job permutation and that of its neighboring job permutations. Here, we still use the same neighborhood structure as in the exploration phase. Firstly, we choose the smallest Δd to be the erosion direction. If the erosion process for that direction does not improve after *MaxUIE* iterations, we say that the direction is blocked. In other words, water flow cannot move in that direction and searching in that direction stops. This is followed by backtracking, in which we restart the search from the local optimal job

permutation using the direction with the next smallest Δd . If all directions for considering the local optimal job permutation are blocked, we call that job permutation fully blocked and move it into the E-list. Then, we do not consider that job permutation in the following clouds or iterations. Otherwise, if there is a direction with improvement when compared with the current local optimal job permutation, we will choose that direction to erode permanently for the local optimal job permutation. The improved local optimal job permutation is updated in the UE-list to continue with performing the erosion process.

The entire process of both exploration and exploitation terminates when the maximum number of allowed clouds or iterations (*MaxCloud*) is reached.

A flow chart of the WFA for solving the PFSP is shown in Figure 4.2. This flow chart is a detailed extension of the flow chart presented in the Chapter 3.

4.3.5 A Numerical Example for Erosion Mechanism

In this section, a numerical example is used to illustrate for implementation of the erosion mechanism in the exploitation phase of WFA. Data of the PFSP instance consists of:

The number of jobs: $n = 5$, and the number of machines: $m = 5$.

The processing time matrix of jobs on machines:

$$P_{ij} = \begin{bmatrix} 2 & 3 & 4 & 9 & 7 \\ 8 & 2 & 6 & 5 & 1 \\ 4 & 2 & 7 & 8 & 5 \\ 2 & 4 & 5 & 6 & 3 \\ 2 & 1 & 3 & 6 & 5 \end{bmatrix}, \quad \text{for} \quad i = 1,..,n \ \text{ and } \ j = 1,..,m.$$

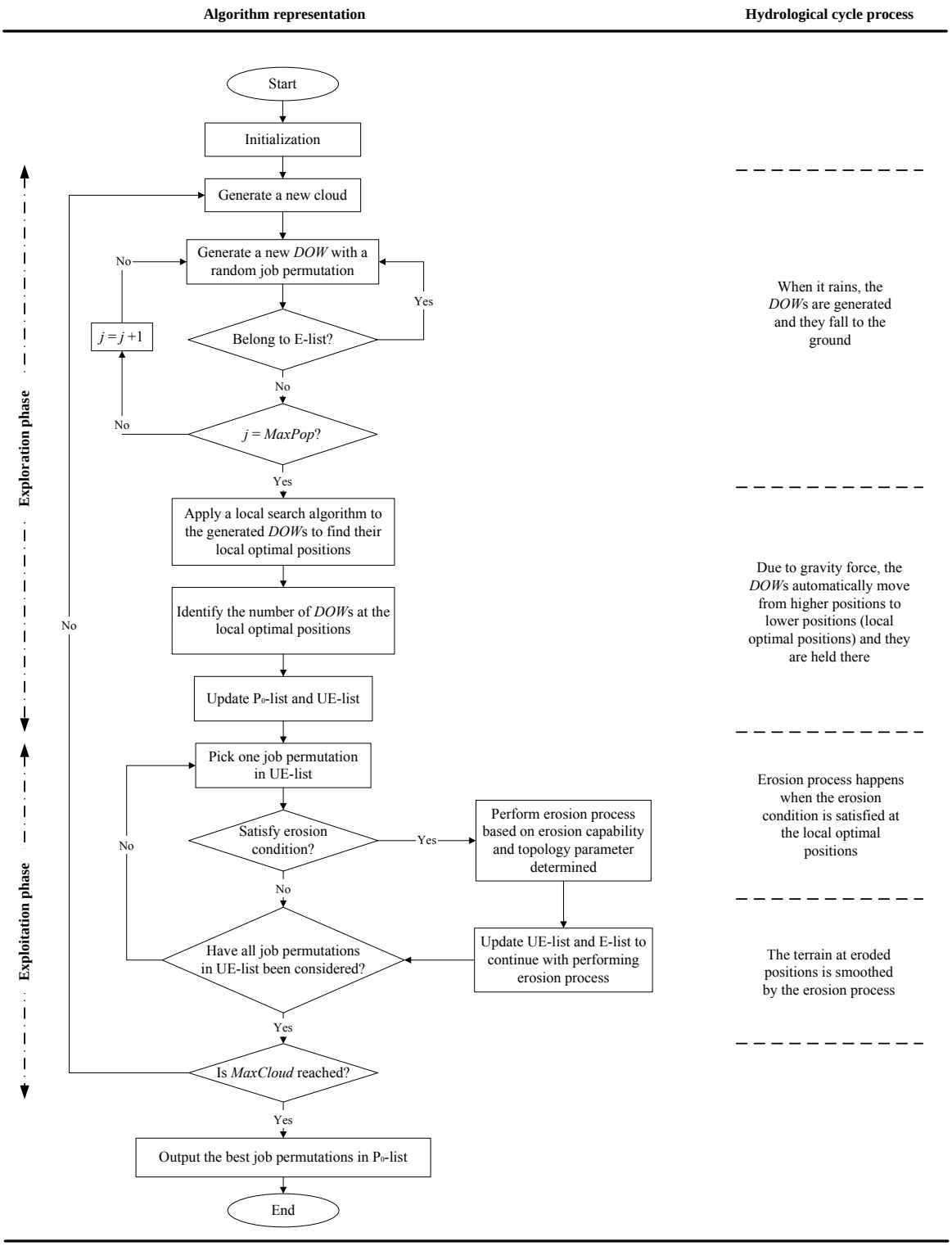


Figure 4.2 Flow Chart of the WFA for the PFSP

For the WFA, the initialization of the parameter values is given as follows: *MaxCloud* = 2, *MaxPop* =7, *MinEro* = 3, *MaxUIE* = 3.

In this illustration, we do not mention to the exploration phase of the WFA, thus we assume that after doing the exploration phase, UE-list contains a set of two local optimal job permutations as follows:

Table 4.1 UE-list of Local Optimal Job Permutations

UE-list	Job permutations	$C_{\max}$	The number of *DOW*s
#1	[2, 1, 4, 3, 5]	47	5
#2	[1, 4, 5, 2, 3]	46	2

We see that only the first local optimal job permutation in the UE-list has satisfied the erosion condition (the number of *DOW*s at the local optimal job permutation is greater than or equals to *MinEro*). Hence, the erosion process will be performed at this job permutation. Firstly, the neighboring job permutations of this local optimal job permutation will be determined and ranked in the descend order with the objective function value $C_{\max}$. In the WFA, only the neighbors of the eroding local optimal job permutation are ranked to consider one after another. For the neighboring job permutations belonging to the search path of the direction chosen, we do not need to rank the job permutations. As a result, ranking neighbors of the eroding local optimal job permutation [2, 1, 4, 3, 5] is shown in Table 4.2.

According to the erosion process of water flow described in the WFA, the smallest topology (or neighboring job permutation) around the eroding local optimal job permutation will be chosen to perform erosion first. Here, we choose the erosion direction

with respect to the neighboring job permutation [1, 2, 4, 3, 5] to perform erosion since its objective value C_{max} = 47 is smallest. In the case that there are many neighbors with the same best objective value, we can choose randomly one of them to perform erosion.

Table 4.2 Possible Erosion Directions at the Local Optimal Job Permutation

Direction	Neighbors	C_{max}
1	[1, 2, 4, 3, 5]	47
2	[2, 1, 4, 5, 3]	47
3	[5, 1, 4, 3, 2]	48
4	[2, 3, 4, 1, 5]	48
5	[3, 1, 4, 2, 5]	49
6	[2, 5, 4, 3, 1]	49
7	[4, 1, 2, 3, 5]	50
8	[2, 1, 5, 3, 4]	50
9	[2, 4, 1, 3, 5]	51
10	[2, 1, 3, 4, 5]	51

Next, we update the eroding local optimal job permutation [2, 1, 4, 3, 5] into the track-list to prevent the *DOW*s move back this position. For the erosion direction chosen, if we cannot find a better job permutation after *MaxUIE* = 3 iterations, we will consider the erosion direction with the next smallest objective value of the neighboring job permutations, i.e., job permutation [2, 1, 4, 5, 3] in Table 4.2. Then, the track-list will be refreshed, and the first erosion direction is known as blocked direction. This procedure is iterated until any improvement job permutation (i.e., the objective value of the job permutation just found is better than that of the eroding local optimal job permutation [2, 1,

4, 3, 5]) is found in a direction. If an improvement job permutation is found, the steepest hill-sliding algorithm is performed to obtain better new local optimal job permutation from the improvement job permutation just found. Otherwise, if there is no improvement through all erosion directions, the eroding local optimal job permutation is known as fully blocked position and recorded into the E-list. Then, the erosion process will consider to the next local optimal job permutation in the UE-list.

In this example, since we may find an improvement job permutation with respect to the first erosion direction [1, 2, 4, 3, 5], this direction will be considered as the permanent erosion direction of the *DOW*s at the local optimal job permutation [2, 1, 4, 3, 5]. The steps to determine the improvement job permutation are shown in Table 4.3.

Table 4.3 Steps of Finding Improvement Job Permutation

Step 2				**Step 3**		
No.	**Neighbors**	$C_{\max}$	**Remarks**	**Neighbors**	$C_{\max}$	**Remarks**
1	[2, 1, 4, 3, 5]	47	in track-list	[4, 1, 2, 3, 5]	50	
2	[4, 2, 1, 3, 5]	50		[2, 4, 1, 3, 5]	51	
3	[3, 2, 4, 1, 5]	48		[3, 4, 2, 1, 5]	46	
4	[5, 2, 4, 3, 1]	49		[5, 4, 2, 3, 1]	49	
5	[1, 4, 2, 3, 5]	47	chosen	[1, 2, 4, 3, 5]	47	in track-list
6	[1, 3, 4, 2, 5]	49		[1, 3, 2, 4, 5]	50	
7	[1, 5, 4, 3, 2]	48		[1, 5, 2, 3, 4]	50	
8	[1, 2, 3, 4, 5]	50		[1, 4, 3, 2, 5]	48	
9	[1, 2, 5, 3, 4]	50		[1, 4, 5, 3, 2]	46	chosen
10	[1, 2, 4, 5, 3]	47		[1, 4, 2, 5, 3]	46	

In particular, we will determine the neighboring job permutations and choose the best job permutation among the neighbors without belonging to the track-list in order to continue searching improvement job permutation. Through steps 2 and 3 in Table 4.3, we find three improvement job permutations. Here, we choose randomly the job permutation [1, 4, 5, 3, 2] to start searching better new local optimal job permutation. When applying the steepest hill-sliding algorithm, we find the local optimal job permutation [1, 4, 3, 5, 2] whose objective value $C_{\max}$ = 45. Then, we update it into the UE-list to continue performing the erosion process as shown in Table 4.4, and also update the eroded local optimal job permutation [2, 1, 4, 3, 5] into the E-list.

Table 4.4 Updating the UE-list

UE-list	**Job permutations**	$C_{\max}$	**The number of *DOW*s**
#1	[1, 4, 3, 5, 2]	45	5
#2	[1, 4, 5, 2, 3]	46	2

This erosion process only terminates when fully blocked state is reached. Then, the next job permutation in the UE-list will be considered to perform the erosion process. If all job permutations in the UE-list are considered, next cloud will be generated to explore new search space. In this example, the entire process of both exploration and exploitation terminates when *MaxCloud* = 2 is reached. Then, the best job permutation found in the P_0-list is displayed as the output of the WFA.

As for the case that the local optimal job permutation is considered as a fully blocked position, we can intuitively observe as shown in Figure 4.3. Then, we cannot find any improvement job permutation through all erosion directions.

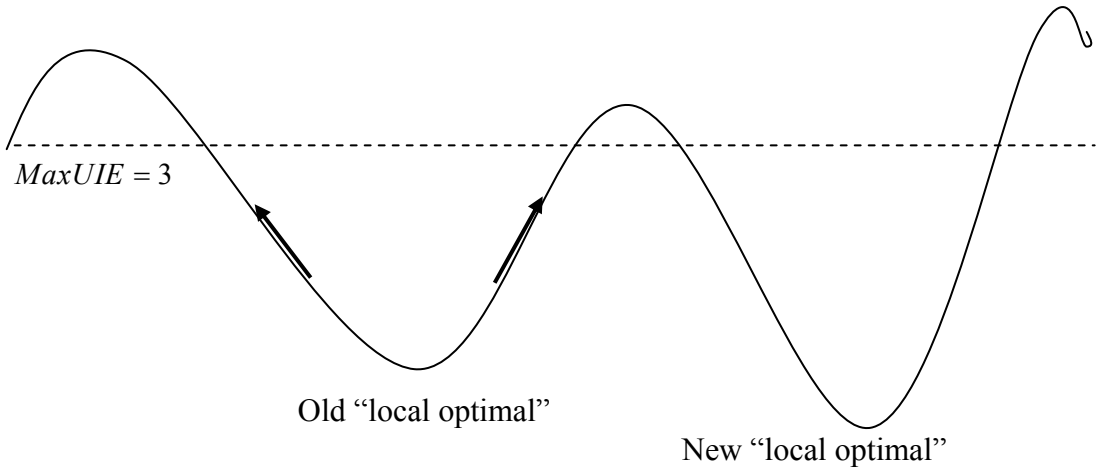


Figure 4.3 The Case of Fully Blocked Position

4.4 Computational Experiments and Comparisons

4.4.1 Benchmark Problem Sets

To evaluate the performance of WFA for the PFSP, we performed experiments for four benchmark problem sets, i.e., that of Carlier, Heller, Reeves and Taillard. These are well-known problem sets in the PFSP taken from the OR Library. In this chapter, a total of 121 instances comprising of 8 instances of Carlier, 2 instances of Heller, 21 instances of Reeves (odd number instances) and 90 instances of Taillard are used. The best known upper bounds of these problem sets from the literature were used to compare with the results obtained by the WFA. Also, the results obtained by Agarwal et al. (2006) and meta-heuristic algorithms in Nagano et al. (2008) were used to compare with our results. Agarwal et al. (2006) performed empirically to select the best parameters, but they did not mention the number of runs of their algorithm for solving the instances; while Nagano et al. (2008) used the design-of-experiment method to tune their parameters and ran 5 independent replicates for each instance.

4.4.2 Platform and Parameters

The WFA has been coded using Visual Basic 6.0, and all experiments were performed on an Intel Centrino Duo 1.60 GHz CPU with 1.5 GB of RAM running on Windows XP Operating System. The computational complexity of the WFA for the PFSP is determined based on the neighborhood structure used and the erosion process of this algorithm. In particular, the WFA used 2-opt neighborhood structure, and the worst possibility of the erosion process is to find through all n directions. Thus, the computational complexity of the WFA may be estimated to be $O(n^3)$.

The choice of reasonable parameters for the WFA was determined by design-of-experiment methods and the values are shown in Table 4.5. With the parameter sets, 5 independent replicates were used for each instance. The average or best results obtained were used to evaluate the performance of the WFA and to compare with other algorithms.

4.4.3 Performance Measure

For comparison of objective values, we used the following average relative percentage increase in objective value:

$$\overline{\Delta} = \frac{\sum_{i=1}^{K}\left(\frac{Heuristic_{sol(i)} - BN_{sol}}{BN_{sol}}\right)}{K} \times 100 \qquad (4.9)$$

where $Heuristic_{sol(i)}$ and BN_{sol} denote the makespan value obtained by the algorithm for the ith replicate and the best known value from the literature, respectively; and K denotes the number of replicates. If we only use the replicate with the best objective value to compute the above $\overline{\Delta}$, then we denote the relative percentage increase in objective value by $\overline{\Delta}_{Best}$.

Table 4.5 Parameter Sets for Benchmark Problem Sets

Benchmark Problem Sets		Value of Parameters			
		MaxCloud	***MaxPop***	***MinEro***	***MaxUIE***
Carlier		5	5	2	5
Heller		5	10	2	5
Reeves	20J-05M	10	10	2	5
	20J-10M	20	20	2	10
	20J-15M	50	20	2	10
	30J-10M	20	20	2	10
	30J-15M	30	10	2	10
	50J-10M	10	10	2	10
	75J-20M	10	10	2	10
Taillard	20J-05M	5	10	2	10
	20J-10M	20	20	2	10
	20J-20M	30	20	2	10
	50J-05M	20	10	2	10
	50J-10M	10	10	2	10
	50J-20M	15	10	2	10
	100J-05M	5	10	2	5
	100J-10M	5	10	2	5
	100J-20M	10	10	2	5

The best known values from the literature are used to be the reference points for the evaluation and comparison of the WFA and other algorithms in this chapter. These values may be the objective values of the optimal solutions of the benchmark instances used, or those of the best solutions found by some algorithm so far. Hence, the comparison results obtained may be negative values if new algorithm finds a better solution. Since we have used the best known values published and summarized in the leading journal papers, the comparison results obtain a high reliability.

4.4.4 Computational Results

From Table 4.6, it can be seen that the WFA outperforms the NEH-ALA although the CPU running time of the WFA in some instances is larger. In particular, 55 out of 90 Taillard's instances solved by the WFA have better results than that solved by the NEH-ALA. In the rest of the instances, the NEH-ALA only performs better than the WFA in 11 instances, which are mainly 100job-5machine instances. For the instances with smaller size, i.e., 20 jobs, we obtained results similar to that of the NEH-ALA, but the CPU running time is smaller. Similar results are also obtained by the WFA when solving Carlier's, Heller's, and Reeves' benchmark problem sets. The WFA obtained the best known upper bound for 10 out of 21 instances of Reeves, while the NEH-ALA only obtained the best known upper bound for 2 instances. Moreover, the WFA has the average relative percentage increase of 0.77%, while it is 1.51% for the NEH-ALA. For the Heller benchmark problem sets, the best known results for these instances are provided in Agarwal et al. (2006), which showed the best makespan value of 136 for the 20job-10machine instance and 516 for the 100job-10machine instance. When applying the WFA to solve the 20job-10machine problem, a new upper bound value of 135 was found.

Also, we compared the results obtained by the WFA with other efficient meta-heuristic algorithms in Nagano et al. (2008) in Table 4.7 and Figure 4.4. From the table and figure, it can be seen that the WFA dominates the performance of the simulated annealing algorithm of Osman and Potts (SAOP), tabu search of Widmer and Hertz (SPIRIT), differential evolutionary method of Onwubolu and Davendra (DE), NEH-ALA of Agarwal et al., and constructive genetic algorithm of Nagano et al. (CGA). When compared with the robust genetic algorithm of Ruiz et al. (GARMA) and CGALS of

Table 4.6 Comparison Results between the WFA and the NEH-ALA

Benchmarks	No. of instances	WFA		NEH-ALA	
		$\overline{\Delta}_{Best}$ (%)	Time (s)	$\overline{\Delta}_{Best}$ (%)	Time (s)
Car11J-5M	1	0	0.17	0	6.6
Car13J-4M	1	0	0.22	0	2.6
Car12J-5M	1	0	0.25	0	8.6
Car14J-4M	1	0	0.23	0	10.7
Car10J-6M	1	0	0.66	0	6.0
Car8J-9M	1	0	0.25	0	6.4
Car7J-7M	1	0	0.11	0	2.8
Car8J-8M	1	0	0.19	0	4.5
Average		***0***		***0***	
Hel20J-10M	1	**-0.74**	23	0	94
Hel100J-10M	1	0	2752	0	2565
Average		***-0.37***		***0***	
Rec20J-5M	3	0	81	0.11	16.5
Rec20J-10M	3	0	218	0.41	34.6
Rec20J-15M	3	0	3732	0.78	49.6
Rec30J-10M	3	0.23	5024	1.54	94.9
Rec30J-15M	3	0.34	4661	1.78	144.3
Rec50J-10M	3	0.27	5223	1.13	423.7
Rec75J-20M	3	4.56	5714	4.81	2876
Average		***0.77***		***1.51***	
Tai20J-5M	10	0	30	0	34.2
Tai20J-10M	10	0	150	0.39	74.1
Tai20J-20M	10	0	854	0.38	152.3
Tai50J-5M	10	0	2173	-0.01	487.8
Tai50J-10M	10	1.42	2837	2.58	983
Tai50J-20M	10	4.30	3877	5.29	2750.3
Tai100J-5M	10	0.29	6842	0.11	3736
Tai100J-10M	10	0.98	7538	1.03	3837
Tai100J-20M	10	4.22	6836	4.97	8853.5
Average		***1.25***		***1.64***	

Table 4.7 Average Relative Percentage Increase over the Best Known Solution for Taillard's Benchmarks Obtained by Meta-heuristic Algorithms

Instances	WFA	SAOP	SPIRIT	GARMA	DE	NEH-ALA	CGA	CGALS
20x5	***0.00***	0.93	4.01	0.29	3.98	1.38	1.33	0.05
20x10	***0.00***	2.59	5.65	0.63	5.86	2.22	2.42	0.19
20x20	***0.00***	2.33	4.84	0.41	4.53	1.78	2.08	0.08
50x5	***0.00***	0.48	1.90	0.06	4.28	0.46	0.32	0.02
50x10	***1.49***	3.34	5.84	1.76	11.48	3.44	3.72	1.65
50x20	4.35	4.47	7.46	***2.62***	14.73	4.66	4.98	2.67
100x5	0.37	0.28	0.93	0.07	4.27	0.46	0.21	***0.02***
100x10	1.11	1.53	2.96	***0.60***	10.42	1.54	1.46	***0.60***
100x20	4.38	4.68	6.26	***2.52***	16.08	4.49	4.52	2.84
Average	**1.30**	**2.29**	**4.43**	**1.00**	**8.40**	**2.27**	**2.34**	**0.90**

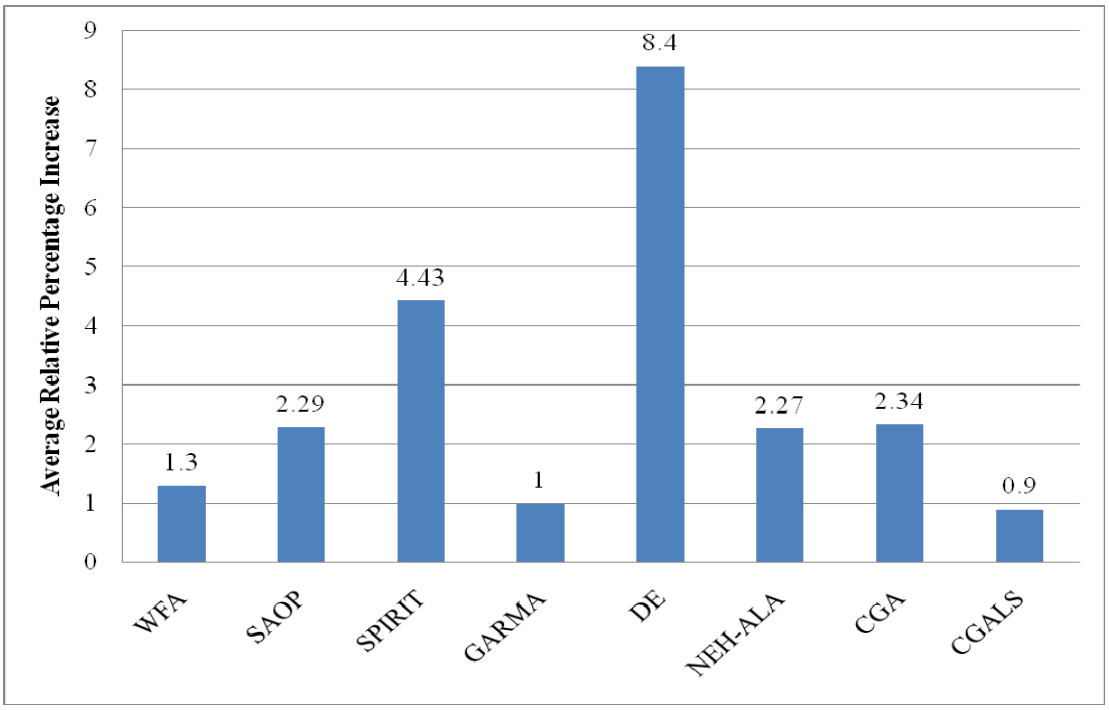


Figure 4.4 Means Plot for Comparing the WFA and Meta-heuristic Algorithms

Nagano et al., the WFA performs best on the small and medium instances (from 20job-5machine to 50job-10machine problems), but for the large instances it performs closer to the average of the group of algorithms, and the SAOP, NEH-ALA and CGA methods than CGALS and GARMA.

4.5 Conclusions

In this chapter, we developed the WFA for solving the PFSP. In the algorithm, only three basic components of the hydrological cycle, i.e., transportation, precipitation, and run-off, were used in the exploration phase. For the exploitation phase, the erosion process with a constant erosion capability was used. The WFA is tested and compared with other meta-heuristic algorithms on the PFSP benchmark problem sets taken from the literature. The results show that the algorithm is able to obtain good solutions to the benchmark problem sets. Also, a new best known solution of a Heller benchmark instance is found by the WFA.

CHAPTER 5

WFA FOR FLEXIBLE FLOW SHOP SCHEDULING

In this chapter, we construct the WFA for solving the flexible flow shop scheduling problem (FFSP), which is one of the NP-hard scheduling problems often encountered in production environment. Here, we investigate the FFSP with limited or unlimited intermediate buffers. A common objective of this problem is to find a production schedule that minimizes the completion time of jobs. Other objectives that we have also considered are minimizing the total weighted flow time of jobs and minimizing the total weighted tardiness time of jobs.

While the proposed WFA is inspired by the hydrological cycle in meteorology and the erosion phenomenon in nature, we have also combined the amount of precipitation and its falling force to form a flexible erosion capability in this algorithm. This helps the erosion process of the WFA to focus on exploiting promising regions strongly. Moreover, to initiate the algorithm, we have used a constructive procedure to obtain a seed job permutation. We have also proposed an improvement procedure for constructing a complete schedule from a job permutation that represents the sequence of jobs in the first

stage of the FFSP. To evaluate the WFA for this scheduling problem, we have used benchmark instances taken from the literature and randomly generated instances of the problem. The computational results demonstrate the efficacy of this algorithm. Also, we have obtained several improved solutions for the benchmark instances using the proposed algorithm. We further illustrate the algorithm's capability for solving problems in practical applications by applying it to a maltose syrup production problem.

The structure of this chapter is organized as follows. In Section 5.1, we introduce the FFSP and its important applications in modern production. A brief literature review of problem classification and solution methods for the problem is also provided in this section. Next, the details of the FFSP with limited or unlimited buffers are described in Section 5.2. A full description of the proposed WFA for the FFSP with intermediate buffers is provided in Section 5.3. An example of the FFSP with limited buffers in maltose syrup production is presented in Section 5.4. Computational experiments and comparisons based on benchmark instances of the FFSP with limited or unlimited buffers, randomly generated instances, and the maltose syrup production problem are shown in Section 5.5. Finally, some conclusions of this chapter are presented in Section 5.6.

5.1 Introduction

In production, flexible flow shop scheduling is one of the well-known NP-hard scheduling problems. In this chapter, we focus on the FFSP with limited intermediate buffers. The problem involves a set of jobs processed through several consecutive operation stages with parallel identical machines in each stage, and there are limited intermediate buffers between consecutive stages. The primary objective of this problem is to find a production

schedule to minimize the completion time of jobs, known as makespan (C_{max}). There are also other important objectives of this problem, such as to minimize the total weighted flow time of jobs and to minimize the total weighted tardiness time of jobs. These objectives help to achieve a high throughput for production. The FFSP with limited buffers has been encountered in both traditional and modern manufacturing systems, such as the electrics manufacturing (Wittrock, 1988), the paper production industry (Sherali et al., 1990), the building industry (Grabowski and Pempera, 2000), the printed circuit board assembly line in electronics industry (Sawik, 2001), and the continuous casting-hot charge rolling production in steel industry (Tang and Xuan, 2006). It is also a special case of the FFSP with unlimited intermediate buffers (Wardono, 2001).

Although many researchers have developed optimization techniques for solving a variety of flow shop scheduling problems, only a few of them dealt with the FFSP with limited buffers. A detailed review of the development of scheduling algorithms, as well as the classification of FFSP, is given in Quadt and Kuhn (2007) and Ribas et al. (2010). Quadt and Kuhn (2007) proposed the taxonomy of *l*-stage flexible flow line scheduling procedures. The taxonomy focuses mainly on heuristic procedures that are split into holistic and decomposition approaches. The decomposition approaches are further classified into stage-oriented decomposition, job-oriented decomposition, and problem-oriented decomposition approaches. Ribas et al. (2010) introduced a new classification for published papers on FFSP. From a production perspective, this classification is based on machine and job characteristics, relevant constraints, and objective functions. From a solution perspective, this classification involves grouping the references into exact approaches, heuristic procedures, hybrid approaches, and simulation/decision support

system procedures. Among these approaches, heuristic and hybrid approaches have recently received considerable attention by many researchers (Ruiz and Vazquez-Rodriguez, 2010). Wardono and Fathi (2004) developed a tabu search algorithm together with a procedure for constructing a complete schedule to solve the FFSP with limited buffers that minimizes job completion time. This algorithm is based on the stage-oriented decomposition approach. Tavakkoli-Moghaddam et al. (2009) proposed a memetic algorithm (MA), which involves a combination of genetic algorithm and nested variable neighborhood search, for solving the flexible flow line scheduling problem with processor blocking and without intermediate buffers. The MA obtained some promising results and it can be considered as an efficient algorithm for solving the FFSP with no available buffer space. However, there is no formal procedure for constructing a complete schedule in the MA. Also, the quality of solutions obtained by these algorithms may not be good for problems with large size.

In this chapter, we construct a water flow algorithm for solving the FFSP with limited or unlimited intermediate buffers. To evaluate the performance of the WFA, we have tested it on many instances of FFSP with intermediate buffers from the literature. Moreover, we also compare the performance of the WFA with that of the tabu search algorithm of Wardono and Fathi (2004) and MA of Tavakkoli-Moghaddam et al. (2009). In addition, we introduce a problem encountered in the maltose syrup production industry and use it to evaluate the efficiency of the WFA for solving problems arising in practical applications.

5.2 FFSP with Intermediate Buffers

The FFSP with limited intermediate buffers is an NP-hard combinatorial optimization problem (Wardono and Fathi, 2004) that may be formulated as follows. A set of N jobs is processed on S consecutive production stages with m_l parallel identical machines in each stage l (l = 1,…,S). There are limited buffers between these consecutive stages (refer to Figure 5.1) and we denote B_l as the capacity of the buffer at stage l. In this problem, jobs have to be processed successively through all S stages. One machine in each stage can only process one job at a time, and each job can only be processed on at most one machine in each stage at the same time. In addition, each job is processed without preemption on one machine in each stage. Moreover, a job can skip one or more stages but is unable to go back to a previous stage. Here, we only consider processing time, weight, and due date of jobs in this problem. Thus, we do not consider other characteristics, such as the breakdown time of machines and set-up time of jobs in this model.

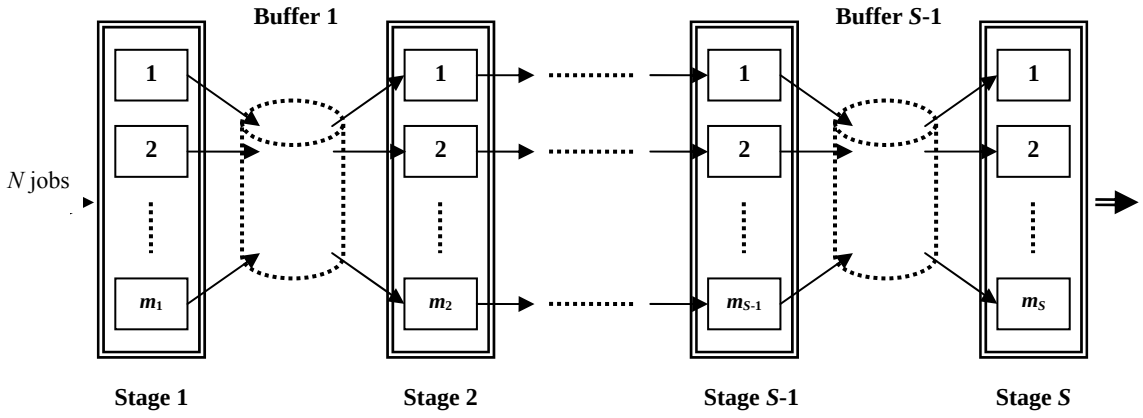


Figure 5.1 The Schematic of FFSP with Limited Intermediate Buffers

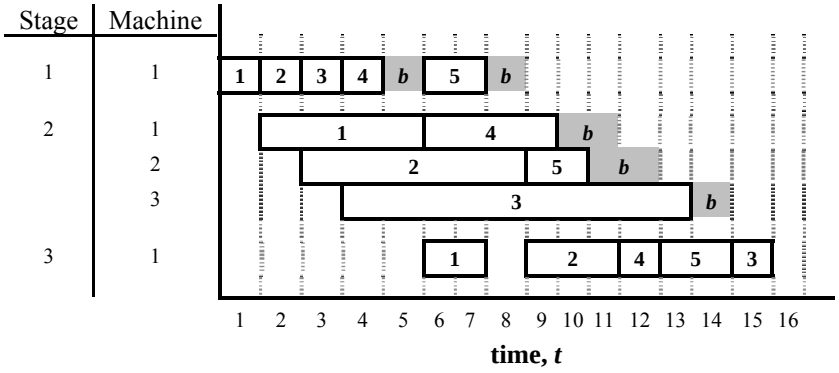
For the FFSP with limited buffers, we can divide it into two cases. The first one is the case of no available buffer space for completed jobs between consecutive stages. It means

that after job *j* is finished by machine *i* in stage *l*, if there is no idle machine in the subsequent stage *l*+1, then job *j* must wait on machine *i* in stage *l* until there is at least an idle machine in stage *l*+1 to start processing job *j* in this stage. The job *j* is known as a blocked job, and the corresponding machine *i* is known as a blocked machine. This case is often encountered in maltose syrup production for the confectionery and sugar industries (Hull, 2010), and the continuous casting-hot charge rolling production for the steel industry (Tang and Xuan, 2006). The second case is the FFSP with finite buffer capacities between consecutive stages. In this second case, after a job *j* is completed on machine *i* in stage *l*, it can either be processed on an available machine in stage *l*+1 or wait in a following buffer if there is no available machine in stage *l*+1. If there is no available capacity in the following buffer, job *j* remains on blocked machine *i* in stage *l* until there is available capacity in the following buffer or an available machine in stage *l*+1. In addition to the two cases mentioned above, we also consider the FFSP with unlimited buffer capacities between consecutive stages. In the FFSP with unlimited intermediate buffers, after a job *j* is completed by machine *i* in stage *l*, the machine *i* is immediately available to process awaiting jobs in the buffers of previous stage *l*-1 even though there is no available machine in stage *l*+1.

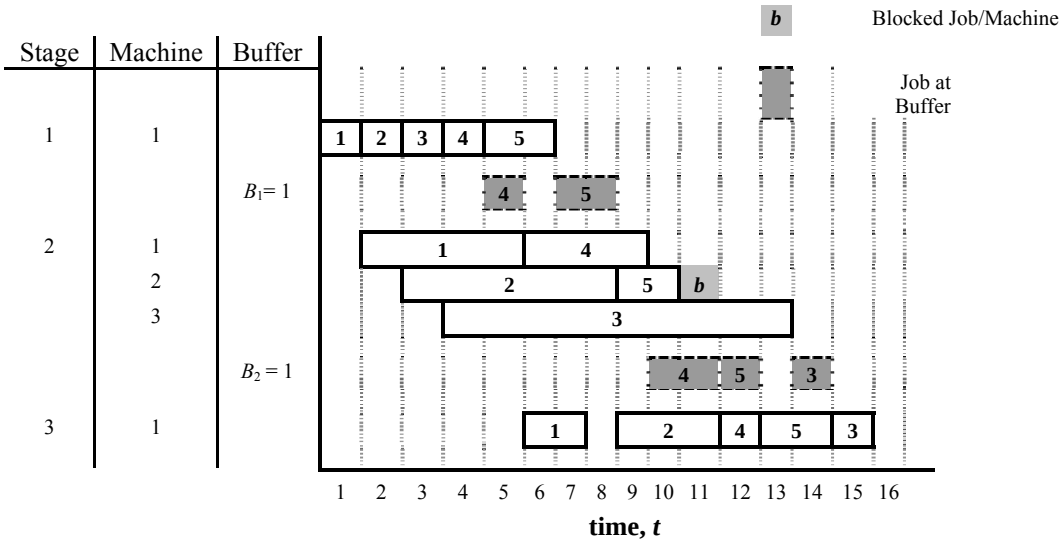
A Gantt chart illustration of the FFSP with intermediate buffers is shown in Figure 5.2. The data for this example is displayed in Table 5.1. In this example, we consider the FFSP with 3 stages in which there is a machine in stage 1 and stage 3, and 3 machines in stage 2. 5 jobs are processed through these stages, in the order of job 1 to job 5 at stage 1. In the first case of the FFSP with limited buffers (Figure 5.2a), when job 4 is completed on machine 1 in stage 1, it has to wait in machine 1 until there is at least an idle machine in

stage 2. Thus, machine 1 in stage 1 is blocked. Hence, job 5 can only be processed on machine 1 in stage 1 from time 5 when machine 1 in stage 2 is available. In the second case of the FFSP with limited buffers (Figure 5.2b), there are buffers with capacity 1 after stage 1 and stage 2, and they are denoted by B_1 and B_2 respectively. With B_1, we can see that machine 1 in stage 1 is not blocked from time 4 to 5 as in the first case. This is because after job 4 is completed on machine 1 in stage 1, it is delivered to buffer 1. Then, job 5 can be processed on machine 1 in stage 1 from time 4. Also, in Figure 5.2b, job 5 is blocked on machine 2 in stage 2 from time 10 to 11, since buffer 2 is full and there is no available machine in stage 3. However, for the FFSP with unlimited buffers (Figure 5.2c), job 5 is not blocked on machine 2 in stage 2 from time 10 to 11 because it is delivered to buffer 2 with infinite capacity. This shows a typical difference between the FFSP with limited buffers and unlimited buffers. With infinite buffer capacities, blocked jobs or machines do not exist in the problem.

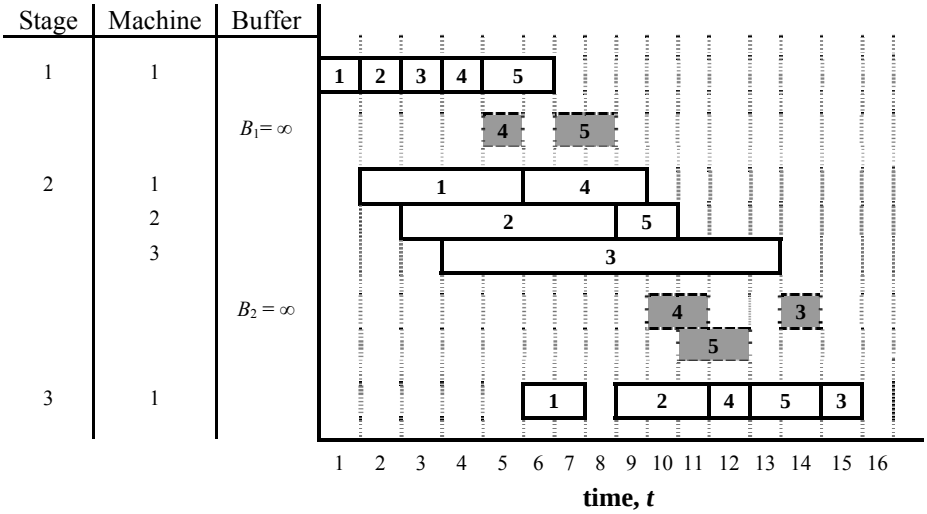
To solve the FFSP with intermediate buffers, we first construct a WFA for solving the FFSP with no available buffer space. Then, we apply the WFA with a modification of the input data to solve the FFSP with finite buffer capacities between consecutive stages. The aim of this modification is to convert the FFSP with finite buffers to one with no available buffer space (McCormick et al., 1989). To do so, we consider a buffer with a capacity of C as a stage with C parallel identical machines in which the processing time of jobs through the machines is zero. We also assume that every job must be processed through all stages, including the buffer stages. We illustrate the conversion process by an example in Table 5.1. As for the FFSP with unlimited buffers, we also perform a modification procedure of constructing a complete schedule that is appropriate with the structure of the problem.



(a) The FFSP with no available buffer space between consecutive stages



(b) The FFSP with finite buffer capacities between consecutive stages



(c) The FFSP with unlimited buffer capacities between consecutive stages

Figure 5.2 A Gantt Chart Illustration of the FFSP with Intermediate Buffers

Table 5.1 An Example of Converting FFSP with Finite Buffers to FFSP with No Available Buffer

Processing time	Stage 1	Stage 2 (buffer stage)	Stage 3	Stage 4 (buffer stage)	Stage 5
Job 1	1	0	4	0	2
Job 2	1	0	6	0	3
Job 3	1	0	10	0	1
Job 4	1	0	4	0	1
Job 5	2	0	2	0	2

5.3 WFA for the FFSP with Intermediate Buffers

In this section, we present the operational mechanism of the WFA for solving the FFSP with intermediate buffers. The proposed algorithm is based on the simulation of spreading of raindrops into many places on the ground, as well as the property of water flow always moving from higher positions to lower positions, and the erosion capability of water flow on the ground.

Firstly, a cloud representing an iteration randomly generates a set of drops of water (*DOW*s) onto some positions on the ground, which represent solutions of the FFSP. Next, due to the gravity force of Earth represented by a heuristic algorithm, the *DOW*s automatically move to local optimal positions. They are held at these positions until the erosion condition is satisfied before performing the erosion process. Then, depending on the amount of precipitation, the falling force of precipitation and soil hardness at the local optimal positions, the erosion process helps the *DOW*s overcome the local optimal positions to find better or global positions. A flow chart of the WFA for the FFSP with intermediate buffers is shown in Figure 5.3. The details of the erosion condition, erosion capability, erosion process, and other operations of the WFA for the FFSP with intermediate buffers are described in the following subsections.

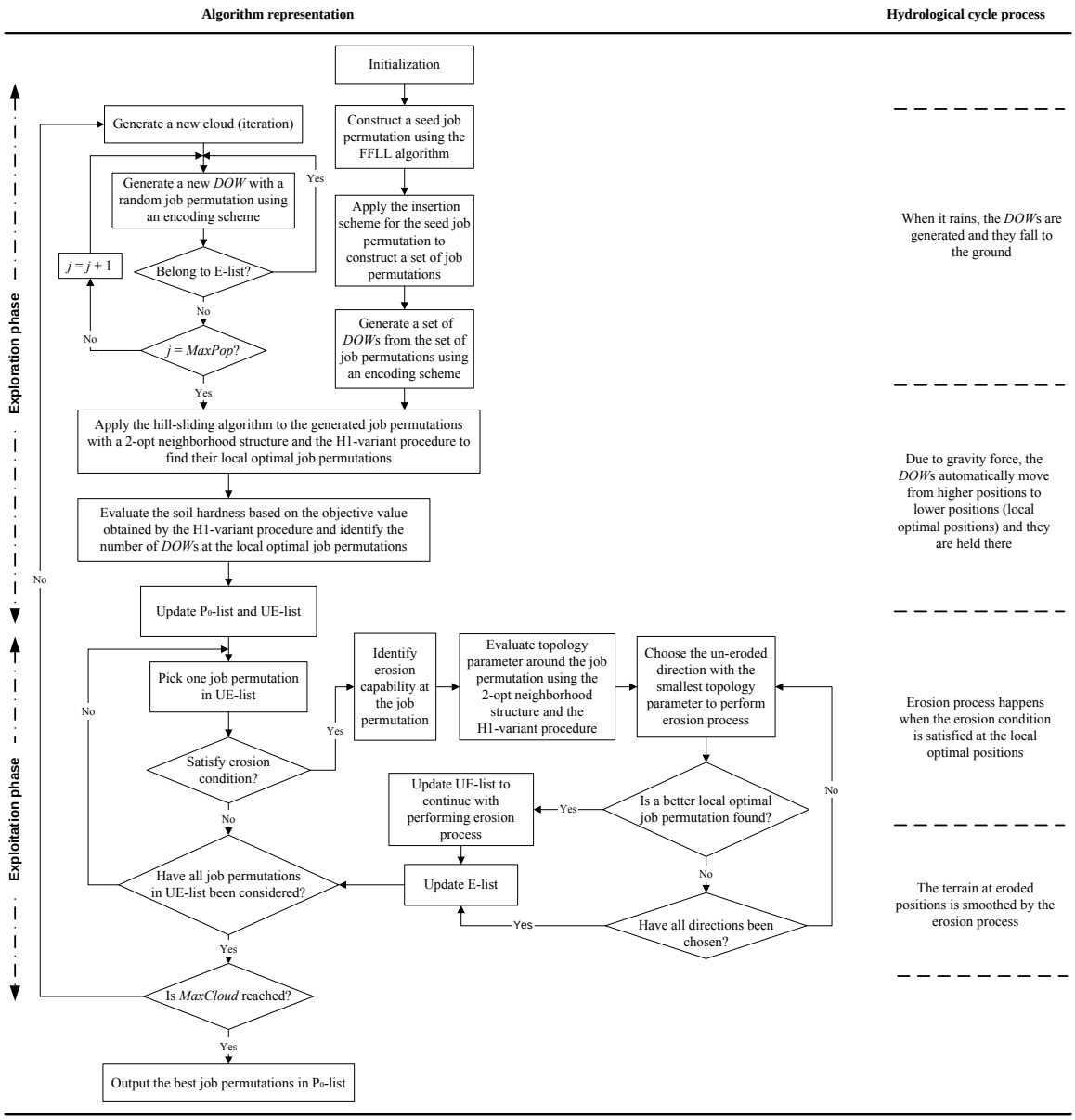


Figure 5.3 Flow Chart of the WFA for the FFSP with Intermediate Buffers

5.3.1 Encoding Scheme

In the FFSP with intermediate buffers, a *DOW* is associated with a job permutation. We consider the job permutations and their single objective value as providing the longitude,

latitude, and altitude information for the position of the *DOW*s on the ground. Given a job permutation $\pi = (\sigma_1, ..., \sigma_N)$, we define:

$$\text{longitude}(\pi) = (\sigma_1, ..., \sigma_{\lfloor \frac{N}{2} \rfloor}) \tag{5.1}$$

$$\text{latitude}(\pi) = (\sigma_{\lfloor \frac{N}{2} \rfloor + 1}, ..., \sigma_N), \tag{5.2}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. The objective value could be the makespan, total weighted flow time of jobs, or total weighted tardiness time of jobs. However, we do not use all the possible objectives at the same time. Depending on the single objective function adopted, the altitude of *DOW* is defined as the corresponding objective value of the given job permutation. Figure 5.4 shows an illustrative example of a *DOW* and its positional vector components for the FFSP with 6 jobs.

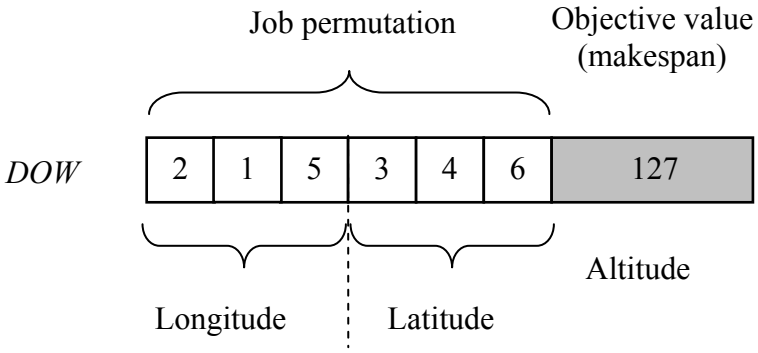


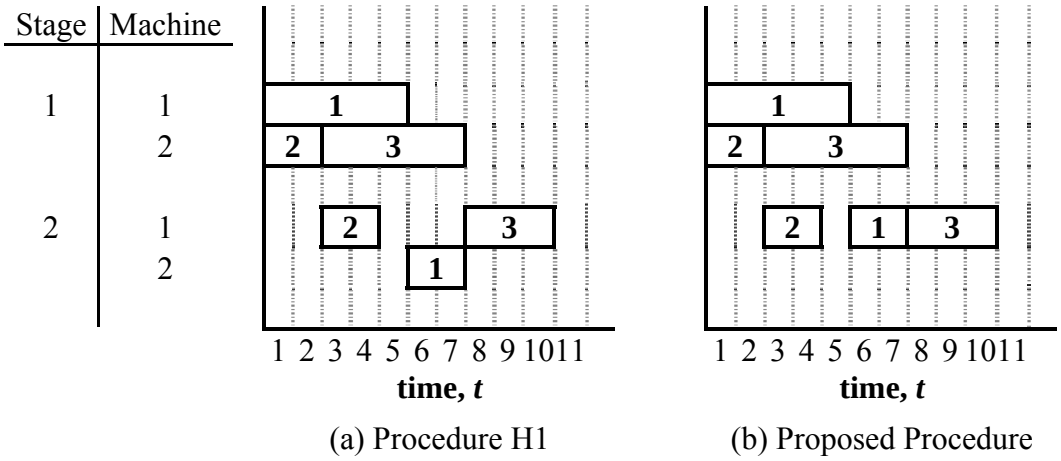
Figure 5.4 An Example of Solution Representation in the WFA for the FFSP

Note that the job permutations considered here are denoted by vectors of size N, which represent the sequence of the given set of jobs performed in the first stage. As such vectors cannot fully determine the schedule of the jobs through all the stages, a procedure for constructing a complete schedule from such vectors is required to obtain the objective value. Here, we propose an improvement over the procedure H1 of Wardono and Fathi

(2004), called the H1-variant procedure, to construct a complete schedule associated with a given job permutation. Our proposed procedure starts from stage 2 and involves choosing a job waiting in the preceding buffer for an available machine in the current stage. The criteria for choosing the job, in order of priority, are job weight, due date, total remaining processing time, processing time at current stage, and length of time at the buffer. Thus, the job with the largest weight has the highest priority to be chosen. If two or more jobs have the same largest weight, then the job with the earliest due date will be chosen first. If there are jobs having the same maximum weight and earliest due date, then the job with maximum total remaining processing time will be chosen. If there is still a tie, the job whose processing time at the current stage is largest will be chosen. Any further ties will be broken by choosing the job that is at the buffer for the longest time. This procedure is unlike the procedure H1 of Wardono and Fathi (2004), in which only the last criterion is used to choose the job to be processed. Moreover, the proposed improvement is applicable to the FFSP with no available buffer space and the FFSP with unlimited buffers. In the former case, we consider the completed jobs which are blocked at the preceding stage as the available jobs waiting in buffer. In the latter case, as the buffer capacities are infinite, the completed jobs are immediately transferred to the buffers to wait for an available machine in the next stage by following the above mentioned rules of choosing the jobs.

In addition, another improvement for assigning available machines based on the job schedule is proposed. The assignment is mainly based on the first available machine (FAM) rule of Wardono and Fathi (2004). However, for the available machines, we give higher priority to the machines that have been used before. An example is shown in Figure

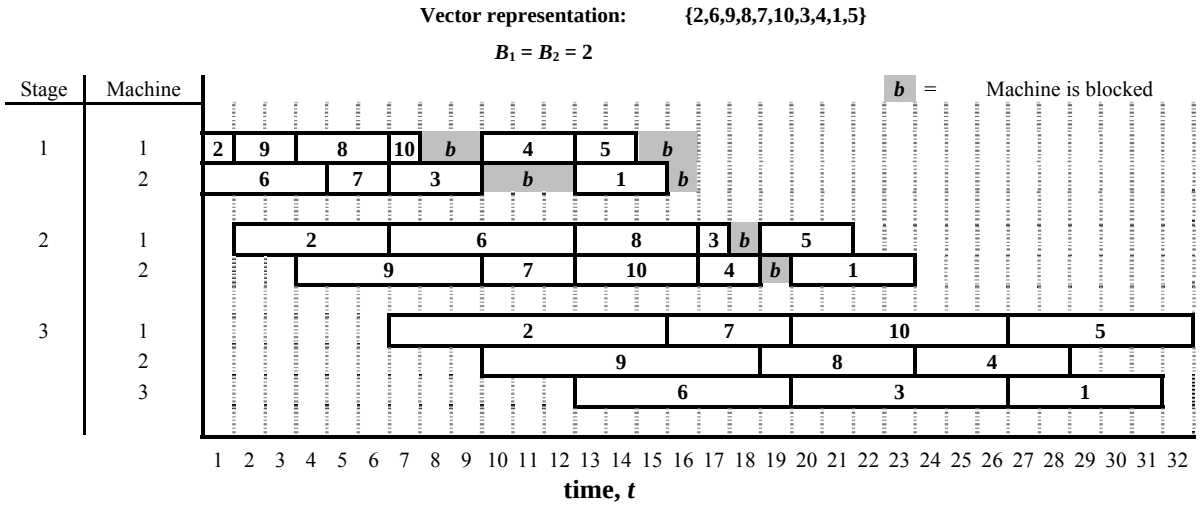
5.5. According to the modified FAM rule, job 1 is processed on machine 1 instead of machine 2 at stage 2, even though machine 2 is available earlier and would have been chosen based on the original FAM rule. This modified FAM rule will help to reduce the number of machines used. Consequently, it has the advantage of possibly reducing the resources used in the design problem.



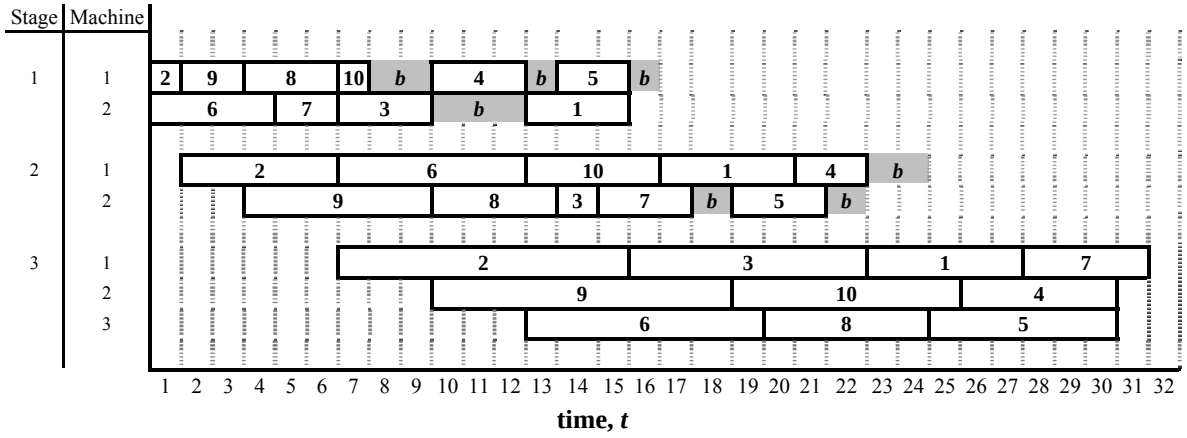
(a) Procedure H1 (b) Proposed Procedure

Figure 5.5 A Comparison Between the FAM Rule and the Modified FAM Rule

Figure 5.6 shows a comparison of the overall performance between the two procedures for an example taken from Wardono and Fathi (2004). In this example, since the weight and due date of jobs are not given, we assume that they are the same for all jobs. Then, choosing the appropriate job for the idle machines is based mainly on the total remaining processing time of available jobs. From Figure 5.6, the completion time of all jobs is at time 31 for the proposed procedure, while the completion time for the schedule of Wardono and Fathi (2004) is at time 32. This shows a better performance of our proposed constructive procedure over that of Wardono and Fathi (2004).



(a) Complete schedule obtained by the procedure H1



(b) Complete schedule obtained by the proposed H1-variant procedure

Figure 5.6 A Comparison Between the Procedure H1 and the H1-Variant Procedure

Based on the constructed complete schedule, the corresponding objective value can then be determined. We thus incorporate the H1-variant procedure of constructing the complete schedule in the determination of the objective value to simplify the computational procedure of the WFA. This simplification is significant because the search space of the problem is now limited to a set of possible permutations of N jobs and we do

not need to look at the number of machines in each stage. Although using the job permutation representation may not cover the global optimal job permutation of the scheduling problem, the best job permutation obtained by the WFA or any algorithm with this representation is very near the global optimal job permutation. Furthermore, the job permutation representation can be more easily integrated into metaheuristic algorithms than the matrix representation, which cover the entire solution space but need expensive computation time to determine whether a solution given in this representation is feasible or not (Wardono and Fathi, 2004).

5.3.2 Memory Lists

Three memory lists of the WFA to support the search for global optimal positions are used in the FFSP with intermediate buffers. Firstly, the best positions list, called the P_0-list, stores the job permutations with minimum makespan, minimum total weighted flow time of jobs, or minimum total weighted tardiness time of jobs. Secondly, the un-eroded list, called the UE-list, is used to store local optimal job permutations which have not been eroded because of not satisfying the erosion condition. Its purpose is to record the potential job permutations for the subsequent erosion process. Thirdly, the eroded list, called the E-list, is used to store eroded local optimal job permutations. Among these lists, the E-list plays an important role of preventing *DOW*s from being regenerated to the eroded job permutations in the subsequent iterations. It would help to reduce the computation time needed by the algorithm. The lists are updated in a similar manner as that in Section 4.3.2.

5.3.3 Exploration Phase

In the first iteration of this phase, we generate a seed job permutation using the flexible flow line loading (FFLL) algorithm (Pinedo, 2005). This is an efficient constructive algorithm for flexible manufacturing systems with cyclic paths. It consists of three phases: the machine allocation phase, the sequencing phase, and the release timing phase. The objective of this algorithm is to minimize the work-in-process so as to reduce blocking probabilities. Then, we use an insertion scheme to generate a set of job permutations for the subsequent erosion process. Here, the insertion scheme is based on the seed job permutation. This scheme is performed by removing a job from its present position and inserting it at a different position, and then shifting the position of jobs between these two positions by a unit accordingly. Although the exploration phase of the WFA is mainly used to explore job permutation search space, the major objective of this phase is to determine potential regions for performing erosion process in the exploitation phase. Hence, we have used the generation of the set of initial seed job permutations to improve the computation time of determining the potential regions. This may also improve the convergence rate of the WFA, although the best job permutations may not come directly from these initial seed job permutations.

In the following iterations of this phase, we no longer use the FFLL algorithm. Instead, only randomly generated job permutations with population size *MaxPop* are used. In all the iterations, after generating a population of job permutations for *DOW*s, a steepest descent hill sliding algorithm is used to search for local optimal job permutations from these initial job permutations. The hill sliding algorithm is similar as that described in Section 4.3.3. In the hill sliding algorithm, a perturbation scheme based on a variant of 2-

opt algorithm is used to construct the neighboring job permutations. The 2-opt neighborhood structure determines the set of all neighboring job permutations that can be obtained from a current job permutation by exchanging positions of two jobs in the current job permutation. In particular, if π' is the job permutation obtained by swapping the positions of two jobs σ_i and σ_j in a job permutation π, we can determine π' by:

$$\begin{aligned} &\pi'[\sigma_i] = \pi[\sigma_j],\ \pi'[\sigma_j] = \pi[\sigma_i], \\ &\pi'[\sigma_k] = \pi[\sigma_k] \ \text{ for } \ k \in N \setminus \{i, j\}, \end{aligned} \tag{5.3}$$

where $\pi[\sigma]$ and $\pi'[\sigma]$ denote the positions of job σ in the job permutation π and its neighboring job permutation π' respectively. The number of neighboring job permutations obtained by the neighborhood structure is $N(N\text{-}1)/2$. The hill sliding algorithm with this 2-opt neighborhood structure is used for all three objective functions considered here.

In general, the exploration phase in the WFA for the FFSP results in a set of local optimal job permutations. They are updated in the UE-list to be considered for performing the erosion process in the next exploitation phase.

5.3.4 Exploitation Phase

5.3.4.1 Erosion Condition and Capability

In the WFA for the FFSP with intermediate buffers, we perform the erosion process based on the amount of precipitation. If the amount of precipitation at some local optimal job permutation increases up to *MinEro* (the minimum number of *DOW*s allowed to start the erosion process), the erosion process would happen at the local optimal job permutation.

In the current WFA, we consider the capability of erosion process based on two main factors, the amount of precipitation and its falling force. In the FFSP with intermediate buffers, the amount of precipitation is represented by the number of *DOW*s at the eroding local optimal job permutation, while its falling force is represented by the objective value at the job permutation. We assume that clouds are at the same altitude, and the falling force of precipitation to lower local optimal positions will cause erosion more easily than that of higher ones, i.e., the erosion capability becomes stronger for local optimal job permutations with larger amount of precipitation and lower objective values. It creates a flexible operation scheme for the erosion capability of *DOW*s in the algorithm, and helps the erosion process focus on exploiting promising regions strongly while ignoring regions with poor performance. In particular, the relationship between these two factors and the control parameter of erosion capability, *MaxUIE* (the maximum number of iterations for the erosion process to move to the next erosion direction if the job permutation is not improved), is expressed as:

$$MaxUIE = \varphi_1 Q(\pi^*) + \varphi_2^{LB/z}, \tag{5.4}$$

where φ_1 and φ_2 are parameters representing the effect of precipitation and its falling force respectively. Also, $Q(\pi^*)$ is the number of *DOW*s at the local optimal job permutation π^*, *LB* is a known lower bound, and *z* is the objective value at the eroding local optimal job permutation. Here, *LB* can be any lower bound obtained from the literature, or by solving a relaxation of the problem. For the case of makespan minimization, we have used the lower bound proposed by Sawik (2000) and Wardono and Fathi (2004) for *LB*. As for the cases of minimizing the total weighted flow time of jobs

and total weighted tardiness time of jobs, we have used the lower bounds in Azizoglu et al. (2001) and Akturk and Yildirim (1998) respectively.

In real-life, the falling force of precipitation does not affect the erosion capability as much as the amount of precipitation. Hence, we formulate the relationship between these two factors and the erosion capability as shown in equation (5.4). While there could be other functions to express the property, from the computational experiments, this has worked well on the scheduling problem. In particular, the effect of the falling force of precipitation is in the range [1, φ_2] by using the function $\varphi_2^{LB/z}$, due to $\frac{LB}{z} \leq 1$. In addition, when running experiments we set φ_2 to vary in [1, 4]. As for the effect of the amount of precipitation, we set φ_1 to vary in [2, 4], since the erosion process is often performed at the local optimal job permutation with $Q(\pi^*) \geq 2$. Then, the falling force of precipitation will have less effect on *MaxUIE* than the amount of precipitation. Although parameters φ_1 and φ_2 may be considered as relative weights, we do not require the sum of the weights to be equal to one.

Based on the preliminary computational experiments by using the design-of-experiment method and setting other parameters to be constant, i.e., *MaxCloud* = 10, *MaxPop* = 20, and *MinEro* = 3, the best values for φ_1 and φ_2 are determined to be 2 and 3 respectively when the WFA is applied to solve the FFSP with intermediate buffers.

5.3.4.2 Erosion Process

The erosion process will be performed when the erosion condition is satisfied. The erosion capability in this process at local optimal job permutations depends on equation (5.4). The strategy of erosion process is based on a topological parameter representing the geographical surface, and whether an erosion direction is blocked.

For the FFSP, the topological parameter Δd is calculated in the same manner as that shown in Section 4.3.4. The backtracking strategy of the erosion process presented in Section 4.3.4 is also used for solving the FFSP by the WFA. Here, we still use the 2-opt neighborhood structure as in the exploration phase of the algorithm.

The entire process for both exploration and exploitation terminates when the maximum number of allowed iterations (*MaxCloud*) is reached.

5.4 An Example of the FFSP in Maltose Syrup Production

In developing countries, confectionery and sugar industries are of great economic importance. The main raw material used for such industries is maltose syrup, and its production is thus crucial for these industries (Pedersen and Vang-Hendriksen, 2001). The production of maltose syrup can be modeled as a FFSP with no available buffer space between consecutive stages for completed jobs. An example of this problem involves nine jobs, with each job handling one type of maltose syrup. They are processed through six consecutive stages representing the six phases in the maltose syrup production process (see Figure 5.7). Here, the process is continuous with connections through pipes and tanks.

The number of parallel identical machines in each stage, processing time of the jobs at these stages, weight of the jobs, and due date of the jobs are shown in Table 5.2.

The jobs processed at the various stages of maltose syrup production not only depend on the busy/idle state of consecutive machines as in the standard FFSP with limited buffers, but also depend on the status of other machines. For example, jobs cannot go to stage 1 until there is at least one idle machine in stage 2, even if the machine in stage 1 is idle. Moreover, since the product is a liquid and stages 2 and 3 are connected directly, the completion time of the jobs in stages 2 and 3 would be the same. As an example, Figure 5.8 illustrates a Gantt chart of the production problem with its data from Table 5.1. Note that job 4 could have been processed at time 3 for the standard FFSP with no available buffer space. However, due to some special requirement in maltose syrup production, job 4 can only be processed from time 7 when there is an idle machine in stage 2. As such, we can consider this as a FFSP with controlled and limited buffers.

Table 5.2 Problem Data for Maltose Syrup Production

Stage	No. of machines	Processing time of jobs (hours)								
		Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9
1	1	6	6	6	6	6	6	6	8	8
2	4	49	49	49	49	49	23	23	11	11
3	1	7	7	7	7	7	7	7	7	7
4	1	2	2	2	2	2	2	2	2	2
5	1	7	5	5	5	5	5	5	7	5
6	1	1	1	1	1	1	1	1	1	1
Due date		120	87	87	172	172	72	72	120	72
Weight		0.10	0.10	0.10	0.05	0.05	0.20	0.20	0.05	0.15

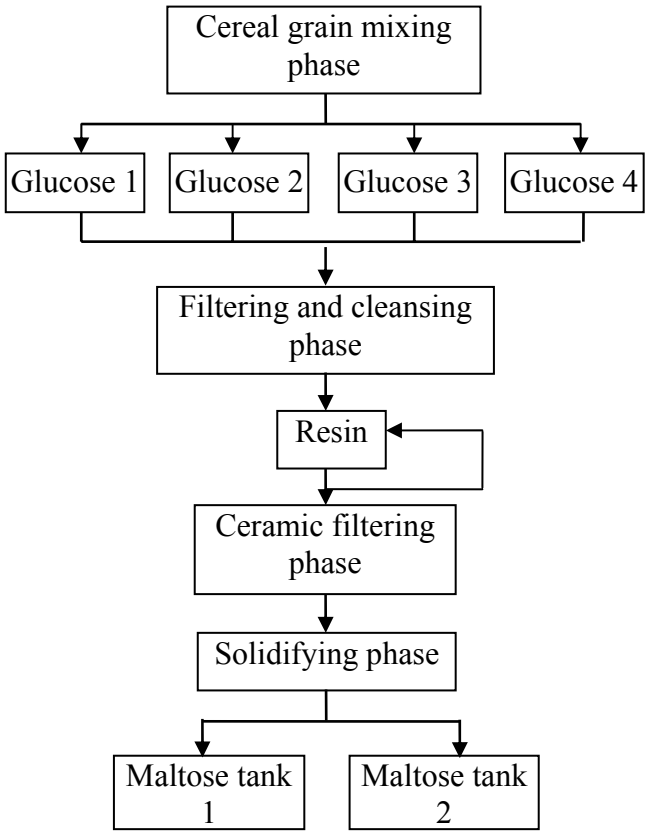


Figure 5.7 Maltose Syrup Production Process

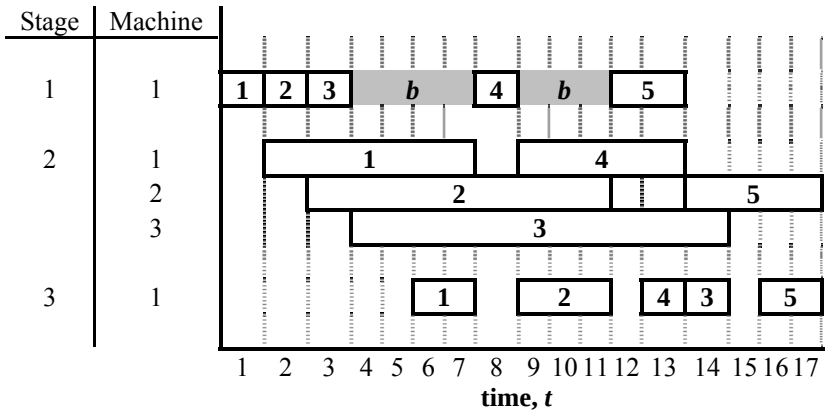


Figure 5.8 Illustration of the FFSP with Controlled and Limited Buffers

Since the special restrictions in the production model only affect the H1-variant procedure of constructing a complete schedule, we do not need to adjust the WFA to solve the maltose syrup production problem. Thus, we consider the restrictions as constraints in the H1-variant procedure for releasing the selected jobs to stage 1 and freeing the machines at stage 2. For the former constraint, the selected jobs can only be processed at stage 1 when at least one machine in stage 1 and one machine in stage 2 are idle. For the latter constraint, the machine finishing a job at stage 2 is free only if the job at stage 3 is also finished. The other rules of selecting and assigning jobs/machines in the H1-variant procedure, as well as the procedure of determining the objective value are unchanged.

5.5 Computational Experiments and Comparisons

In this section, we present the results of computational experiments carried out on the benchmark instances of Wittrock (1988), the randomly generated instances based on Wardono and Fathi (2004), the randomly generated instances based on Tavakkoli-Moghaddam et al. (2009), and an instance of the maltose syrup production problem mentioned in the previous section. These experiments are used to evaluate the performance of the WFA for solving the FFSP with limited and unlimited buffers.

5.5.1 Benchmark Instances and Randomly Generated Instances

Here, the benchmark instances of Wittrock (1988) shown in Table 5.3 are modified in a similar manner as Wardono and Fathi (2004). Thus, where applicable, the transport time (one minute) for the Wittrock instances is added to the processing time of the jobs at stages 2 and 3 respectively. In these instances, the number of machines at stages 1, 2, and

3 are 2, 3, and 3 respectively. When solving the instances with finite buffers, we set the buffer capacity to 3 for all stages.

For the instances generated based on Wardono and Fathi (2004), we use the procedure of generating the type II dataset in that paper. These instances, called the TS instances, were constructed with the number of jobs N = 20, 30, 40, 50; the number of stages S = 2, 3, 4; and the number of machines m_l = 2, 4, 6 for all stages l. For each job, its processing time is generated randomly as an integer from a uniform distribution [1, 100]. For each set of parameter values, we construct and solve five instances. When solving the instances with finite buffers, we set the buffer capacity to 1 for all stages. We compare the results obtained by the WFA to the lower bounds proposed by Wardono and Fathi (2004). The results for the Wittrock benchmark instances and the type II dataset obtained by the tabu search algorithm of Wardono and Fathi (2004) are used to compare against the results obtained by the WFA. In the comparison, the tabu search algorithms are called TS-H1 and TS-Z3 for the FFSP with limited and unlimited buffers respectively.

Table 5.3 Problem Data for the Instances in Wittrock (1988)

Job type	Processing time (min)			Production requirement of each instance					
	Stage 1	Stage 2	Stage 3	1	2	3	4	5	6
A	39	11	14	12	-	-	-	-	-
B	13	28	54	1	-	-	-	-	-
C	22	56	60	26	-	-	14	23	20
D	234	39	0	-	-	-	2	-	-
E	39	25	80	-	6	7	4	-	1
F	13	70	54	-	14	20	16	-	-
G	143	66	0	1	4	-	-	3	1
H	0	28	14	7	-	-	-	-	-
I	26	39	74	-	6	-	-	-	5
J	18	59	34	-	4	-	-	-	-
K	22	70	40	4	-	-	-	-	-
L	13	70	54	-	4	-	-	-	-
M	61	46	34	-	-	11	-	14	3
	Total number of jobs			51	38	38	36	40	30

The instances generated based on Tavakkoli-Moghaddam et al. (2009) are called the MA instances. They are constructed with the number of jobs N = 10, 20, 30, 40; and the number of stages S = 2, 3, 4, 5, 6. Here, the number of machines in every stage is the same, and the total number of machines in all stages is equal to twice the number of stages. For each job, its integer processing time is generated by a uniform distribution [1, 10]. With the combination of the above parameter values, we generate 20 different problem instances. We compare the results obtained by the WFA to the lower bounds proposed by Sawik (2000), as well as to the results obtained by Tavakkoli-Moghaddam et al. (2009), which describes an algorithm for solving the FFSP with no available buffer space.

In addition to comparing with the results reported in Wardono and Fathi (2004) and Tavakkoli-Moghaddam et al. (2009), we perform computational experiments on the TS and MA instances with the same CPU and platform for all algorithms, i.e. WFA, TS-H1/Z3, and MA. As the instances used in Wardono and Fathi (2004) and Tavakkoli-Moghaddam et al. (2009) do not include the weight and due date of jobs, we also do not consider these job characteristics in the randomly generated instances, even though the WFA is capable of solving the FFSP with given weight and due date of jobs.

5.5.2 Platform and Parameters

The WFA has been coded using Visual Basic 6.0, and all experiments have been performed on an Intel Centrino Duo 1.60 GHz CPU with 1.5 GB of RAM running on Windows XP Operating System. The computational complexity of the WFA for the FFSP is determined based on the neighborhood structure used and the erosion process of this algorithm. In particular, the WFA used 2-opt neighborhood structure, and the worst

possibility of the erosion process is to find for all n directions. Hence, the computational complexity of the WFA is estimated to be $O(n^3)$.

The choice of reasonable parameters for the WFA is determined by design-of-experiment methods. When implementing the design-of-experiment method, we may use the independent level of values for the parameters, or the dependent ratio among the parameters, e.g., the ratio of *MaxPop* and *MinEro*, to determine which values are best for solving the scheduling problem. Here, we used the independent level of values for the parameters since we have solved many types of FFSP data structures this way. The independent levels of values for parameters were used as follows: *MaxCloud* = 5, 10, 15, 20; *MaxPop* = 5, 10, 15, 20; and *MinEro* = 2, 3, 4, 5. From the preliminary results, the best parameter sets are summarized in Table 5.4. With these parameter sets, 20 independent replicates are used for each instance of Wittrock (1988), the maltose syrup production example, and the MA instances as in the case of Tavakkoli-Moghaddam et al. (2009). However, we only use 1 replicate for the TS instances as in the case of Wardono and Fathi (2004). Here, Wardono and Fathi (2004), and Tavakkoli-Moghaddam et al. (2009) performed a preliminary study to determine the best parameter values for TS-H1/Z3, and MA respectively.

Table 5.4 Parameter Sets for Benchmark Instances

Instances	Value of parameters		
	MaxCloud	*MaxPop*	*MinEro*
Modified Wittrock			
Instance 1	10	20	3
Instance 2, 3, 5	10	15	3
Instance 4, 6	10	10	4
Wardono & Fathi	10	20	3
Tavakkoli-Moghaddam	10	20	3
Maltose syrup production	5	10	5

5.5.3 Performance Measures

For a comparison of objective values, we have used the following average relative percentage increase in objective value:

$$\overline{\Delta} = \frac{\sum_{i=1}^{K}\left(\frac{Heuristic_{sol(i)} - LB_{sol}}{LB_{sol}}\right)}{K} \times 100. \qquad (5.5)$$

Here, $Heuristic_{sol(i)}$ and LB_{sol} denote the objective value obtained by the algorithm for the *i*th replicate and the lower bound value or the best known objective value from the literature respectively, while K denotes the number of replicates. If we only use the replicate with the best objective value to compute $\overline{\Delta}$, the relative percentage increase in objective value is denoted by $\overline{\Delta}_{Best}$. The smaller the value of $\overline{\Delta}$, the better the performance of the algorithm. In this chapter, the lower bounds are used in all instances and the optimal solution is only used in the maltose syrup production problem, so that $\overline{\Delta} = 0$ implies that the algorithm has obtained the optimal solution.

The lower bound value or the best known objective value from the literature is used as the reference values for the evaluation and comparison of the WFA and other algorithms in this chapter. The best known objective value may be obtained from the optimal solutions of the benchmark instances used, or the best solutions found by some algorithm so far. Hence, the comparison results may be negative values if the proposed algorithm finds a better solution.

For demonstrating the effect of problem size on the speed of different algorithms, we use a CPU time ratio measure, denoted simply as *ratio*. This is the ratio of CPU time for

solving an instance under consideration to CPU time for solving the instance with the smallest size, using the same number of iterations. If the value of *ratio* of an algorithm increases quickly with respect to an increase in the problem size, it means that the speed of the algorithm depends significantly on the problem size.

5.5.4 Computational Results

To compare the algorithms, we performed computational experiments on two sets of generated TS and MA instances, using the same CPU and platform mentioned in Section 5.5.2 for all the algorithms, i.e., WFA, TS-H1/Z3, and MA. Instead of using the maximum number of iterations in the experiments, CPU time limits with respect to the number of jobs in the instances are imposed on all the algorithms. Table 5.5 shows the results of the WFA, TS-H1/Z3, and MA on the TS instances. Here, we use the lower bound obtained by Wardono and Fathi (2004) for LB_{sol} in equation (5.5). The CPU time limits for the case of FFSP with unlimited buffers are equal to one-sixth of the CPU time limits for the other cases. This is because the procedure for constructing a complete schedule in the FFSP with unlimited buffers requires a shorter computation time. As the MA is only designed for the FFSP with no available buffer, we only run experiments for MA on TS instances with $B_l = 0$. Table 5.6 shows the results of the WFA, TS-H1, and MA on the MA instances for the case of FFSP with no available buffer. In this table, SD ($C_{\max}$) and SDOFV represent the standard deviation of objective values obtained by the WFA/TS-H1 and the MA respectively. Here, we use the lower bound obtained by Sawik (2000) for LB_{sol} in equation (5.5). From the results in Tables 5.5 and 5.6, we can see that the solution quality obtained by the WFA outperforms the other algorithms for most instances. In a few

Table 5.5 Comparison Results of WFA, TS-H1/Z3, and MA with CPU Time Limit for the TS Instances

Test problem		CPU time limit (s)	WFA			TS-H1/Z3			MA
			$\overline{\Delta}$			$\overline{\Delta}$			$\overline{\Delta}$
No.	$N \times S \times m_l$		$B_l = 0$	$B_l = 1$	$B_l = \infty$	$B_l = 0$	$B_l = 1$	$B_l = \infty$	$B_l = 0$
1	20× 2 × 2	300 if $B_l = 0, 1$; 50 if $B_l = \infty$	0.16	0.04	0.04	0.52	0.04	0.10	5.09
2	20 × 2 × 4		1.14	0.71	1.11	2.51	1.79	2.69	18.58
3	20 × 2 × 6		4.02	4.01	5.39	7.49	6.49	8.11	26.74
4	20 × 3 × 2		0.57	0.25	0.21	2.49	0.29	0.67	8.75
5	20 × 3 × 4		2.04	1.37	1.71	6.37	4.04	2.81	14.94
6	20 × 3 × 6		1.15	0.99	1.61	4.74	2.27	2.96	19.25
7	20 × 4 × 2		4.25	1.99	1.89	4.80	1.94	1.29	17.45
8	20 × 4 × 4		7.48	7.10	6.82	11.81	8.96	7.03	24.77
9	20 × 4 × 6		8.00	6.36	7.60	12.29	10.33	9.09	22.32
10	30 × 2 × 2	600 if $B_l = 0, 1$; 100 if $B_l = \infty$	0.83	0.07	0.15	1.63	0.27	0.09	5.57
11	30 × 2 × 4		1.77	0.78	0.94	2.69	2.38	1.30	9.28
12	30 × 2 × 6		3.16	2.49	2.52	6.67	3.10	3.78	17.69
13	30 × 3 × 2		1.95	0.09	0.06	2.03	0.04	0.09	15.33
14	30 × 3 × 4		3.72	1.56	1.35	5.19	2.33	1.49	22.79
15	30 × 3 × 6		5.36	5.00	4.16	11.16	8.34	5.97	24.65
16	30 × 4 × 2		4.48	1.57	1.42	4.77	1.97	1.42	21.00
17	30 × 4 × 4		7.18	6.10	6.27	9.01	7.34	6.17	31.51
18	30 × 4 × 6		10.69	9.88	9.61	10.63	9.01	8.68	30.34
19	40 × 2 × 2	1200 if $B_l = 0, 1$; 200 if $B_l = \infty$	0.68	0.05	0.15	1.04	0.12	0.13	4.44
20	40 × 2 × 4		2.04	0.71	0.71	2.24	0.95	1.12	11.79
21	40 × 2 × 6		2.90	1.68	2.04	3.67	2.74	2.02	18.81
22	40 × 3 × 2		5.22	0.37	0.31	6.62	0.29	0.27	22.02
23	40 × 3 × 4		4.57	1.68	1.94	4.64	1.93	1.42	26.49
24	40 × 3 × 6		6.35	4.86	4.51	6.95	5.86	4.24	32.24
25	40 × 4 × 2		6.57	1.51	1.34	7.27	1.67	0.95	32.09
26	40 × 4 × 4		6.17	4.11	2.92	7.84	4.69	2.53	44.95
27	40 × 4 × 6		8.27	7.36	7.68	8.71	7.34	6.09	40.84
28	50 × 2 × 2	1800 if $B_l = 0, 1$; 300 if $B_l = \infty$	0.32	0.02	0.04	1.49	0.02	0.07	10.52
29	50 × 2 × 4		0.99	0.29	0.26	1.58	0.86	0.42	14.23
30	50 × 2 × 6		2.07	0.98	0.98	3.06	1.07	2.10	19.76
31	50 × 3 × 2		4.36	0.43	0.34	4.69	0.38	0.31	19.87
32	50 × 3 × 4		3.68	1.64	0.86	3.37	1.39	1.06	27.98
33	50 × 3 × 6		6.39	3.78	3.40	7.72	4.26	3.71	34.31
34	50 × 4 × 2		9.39	1.71	0.69	10.01	2.04	0.68	33.63
35	50 × 4 × 4		7.46	3.97	2.26	9.10	4.12	2.43	47.71
36	50 × 4 × 6		6.91	5.04	3.02	7.39	6.26	3.20	48.98
	Average		**4.23**	**2.51**	**2.40**	**5.67**	**3.25**	**2.68**	**22.96**

Table 5.6 Comparison Results of WFA, TS-H1, and MA with CPU Time Limit for the MA Instances

Test problem		CPU time limit (s)	WFA		TS-H1		MA	
No.	$N \times S \times m_l$		$\overline{\Delta}$	SD (C_{max})	$\overline{\Delta}$	SD (C_{max})	$\overline{\Delta}$	SDOFV
1	10 × 2 × 2	120	0.00	0.00	0.00	0.00	0.00	0.00
2	10 × 3 × 2		5.00	0.00	5.00	0.00	8.87	0.60
3	10 × 4 × 2		21.35	0.00	21.35	0.00	24.83	0.83
4	10 × 5 × 2		25.30	0.00	25.30	0.00	33.98	0.82
5	10 × 6 × 2		33.33	0.00	33.33	0.00	40.25	1.51
6	20 × 2 × 2	300	0.00	0.00	0.00	0.00	4.15	0.69
7	20 × 3 × 2		6.75	0.37	6.67	0.41	20.44	1.09
8	20 × 4 × 2		17.26	0.44	17.61	0.51	32.39	0.52
9	20 × 5 × 2		17.90	0.64	18.23	0.66	34.11	2.62
10	20 × 6 × 2		21.97	0.51	22.55	0.51	43.65	2.01
11	30 × 2 × 2	600	1.92	0.44	3.90	0.60	11.98	0.45
12	30 × 3 × 2		1.95	0.47	3.30	0.60	17.62	0.62
13	30 × 4 × 2		11.80	1.03	13.33	1.03	32.08	1.04
14	30 × 5 × 2		20.22	0.68	21.09	0.75	45.76	3.51
15	30 × 6 × 2		22.30	0.79	23.55	0.89	53.11	1.94
16	40 × 2 × 2	1200	0.42	0.00	0.46	0.22	7.85	0.83
17	40 × 3 × 2		5.10	0.67	6.91	1.02	20.60	1.84
18	40 × 4 × 2		11.12	1.00	13.18	1.15	38.26	1.42
19	40 × 5 × 2		15.85	0.83	17.67	0.93	47.71	1.04
20	40 × 6 × 2		13.90	1.09	15.51	1.13	45.20	2.01
	Average		**12.67**	**0.45**	**13.45**	**0.52**	**28.14**	**1.27**

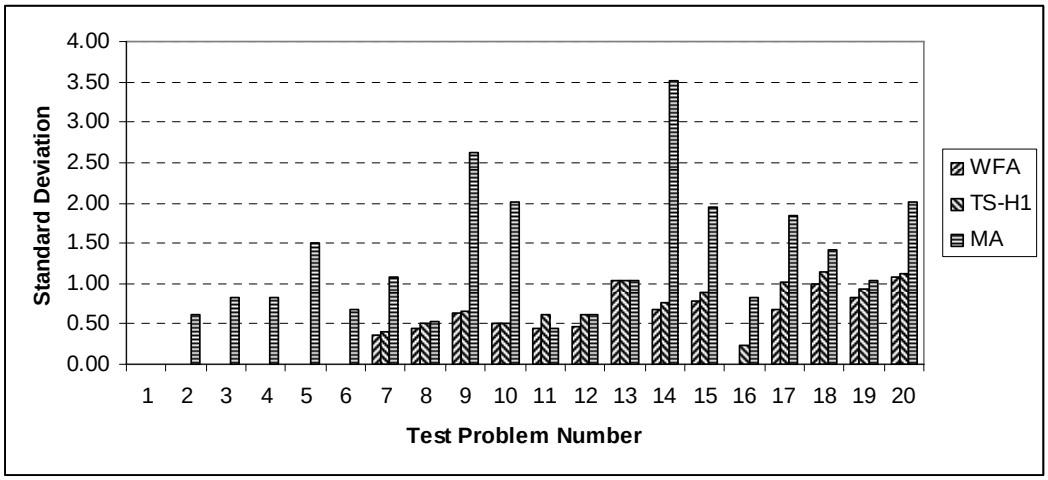


Figure 5.9 Standard Deviation of the Objective Values Obtained by the WFA, TS-H1 and MA

instances, TS-H1/Z3 obtained slightly better results than WFA, especially in the case of FFSP with unlimited buffers. However, from Figure 5.9, we can see that the standard deviation values of WFA are smaller than that of the other algorithms in all instances. This shows the robustness of the WFA when compared to the other algorithms.

In addition, we also investigate the tendency of the algorithms, i.e., WFA, TS-H1 and MA, to find better solutions over computation time. The computational experiments are carried out on the MA instance with 40 jobs, 6 stages, and 2 machines at each stage, as well as CPU time limit of 1200 seconds for all algorithms. From Figure 5.10, we see that the WFA may find a very good solution more quickly than TS-H1 and MA. Then, the WFA improves this solution slowly to obtain a better solution or global optimal solution. The final best solution obtained by the WFA is better than that of TS-H1 and MA.

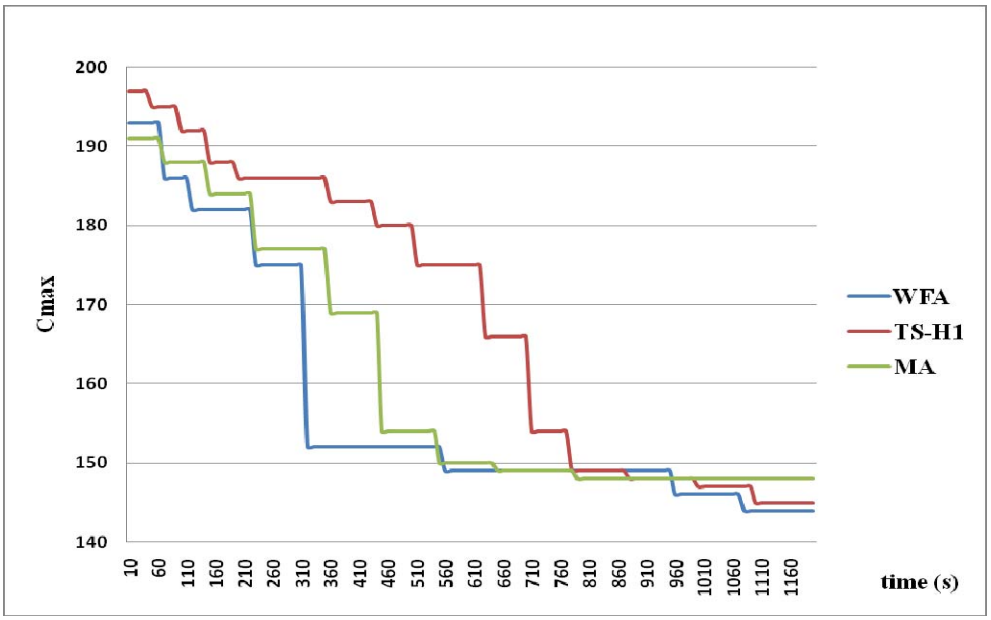


Figure 5.10 Trajectory of Solution Improvement of the WFA, TS-H1, and MA

A further comparison between the results of the WFA using the maximum number of allowed iterations and the results of the tabu search algorithms reported in Wardono and Fathi (2004) on the randomly generated TS instances is performed in Table 5.7. We use the lower bound obtained by Wardono and Fathi (2004) for LB_{sol} in equation (5.5). From Table 5.7, we can see that the WFA is more efficient than TS-H1/Z3 when solving all three FFSP models. For the FFSP with no available buffer, the average relative percentage increase of the WFA is 3.9% less than that of TS-H1. For the FFSP with finite buffers, the average relative percentage increase of the WFA is 2.22% less than that of TS-H1. For the FFSP with unlimited buffers, the average relative percentage increase of the WFA is 1.7% less than that of TS-Z3. However, for some of the generated instances, the CPU time ratio of the WFA is more than that of TS-H1/Z3.

A further comparison between the results of the WFA using the maximum number of allowed iterations and the MA's results reported in Tavakkoli-Moghaddam et al. (2009) for solving the randomly generated MA instances is performed in Table 5.8. We use the lower bound obtained by Sawik (2000) for LB_{sol} in equation (5.5). From the results in Table 5.8, we can see that the WFA outperforms MA in solution quality for all the instances. The overall average relative percentage increase of the WFA is 14.63% less than that of MA. The WFA is also more robust than MA as its average standard deviation is less than that of MA. However, the CPU time ratio of WFA is larger than that of MA from instance 12 onwards.

Table 5.7 Comparison Results between WFA and TS-H1/Z3 on the Randomly Generated TS Instances

Test Problem		WFA						TS-H1/Z3					
		$\overline{\Delta}$			*ratio*			$\overline{\Delta}$			*ratio*		
No.	$N\times S\times m_l$	$B_l=0$	$B_l=1$	$B_l=\infty$	$B_l=0$	$B_l=1$	$B_l=\infty$	$B_l=0$	$B_l=1$	$B_l=\infty$	$B_l=0$	$B_l=1$	$B_l=\infty$
1	20× 2 × 2	0.31	0.10	0.10	1.00	1.00	1.00	1.45	0.47	0.15	1.00	1.00	1.00
2	20 × 2 × 4	1.00	0.20	0.20	1.54	1.76	1.58	4.78	2.41	1.99	1.09	1.50	1.91
3	20 × 2 × 6	6.56	5.40	4.82	2.16	2.47	2.56	8.72	7.79	8.00	1.59	1.43	2.18
4	20 × 3 × 2	0.08	0.08	0.08	2.57	2.64	2.68	5.85	0.79	0.49	2.62	1.81	2.67
5	20 × 3 × 4	0.43	0.14	0.14	2.91	3.65	2.61	10.24	9.16	8.11	2.67	2.12	2.55
6	20 × 3 × 6	0.00	0.00	0.00	3.46	5.04	3.53	1.52	1.58	1.44	2.23	2.13	2.39
7	20 × 4 × 2	3.91	0.00	0.00	4.08	6.95	5.05	10.56	3.18	2.81	3.04	4.84	3.85
8	20 × 4 × 4	2.27	1.20	1.20	4.68	6.64	4.37	11.88	10.26	10.24	3.41	4.87	5.03
9	20 × 4 × 6	5.29	5.29	5.29	5.27	8.06	5.97	10.35	10.04	9.17	3.80	3.00	4.33
10	30 × 2 × 2	0.15	0.00	0.00	4.55	3.84	3.26	3.84	0.33	0.19	3.34	3.04	2.61
11	30 × 2 × 4	0.43	0.43	0.14	4.93	4.22	4.91	3.09	1.30	0.66	3.46	3.05	3.70
12	30 × 2 × 6	1.21	1.21	0.78	5.22	5.41	8.75	5.13	3.37	2.76	4.23	4.13	6.91
13	30 × 3 × 2	2.29	0.00	0.00	6.13	7.14	9.69	7.93	0.80	0.46	4.86	5.71	5.39
14	30 × 3 × 4	3.86	1.49	0.53	9.72	10.25	12.41	6.38	4.68	3.23	8.03	6.86	8.12
15	30 × 3 × 6	4.95	4.26	4.26	10.55	14.49	15.78	7.30	6.01	4.95	7.89	8.99	9.82
16	30 × 4 × 2	1.31	0.24	0.00	11.03	11.64	11.51	11.51	3.38	2.31	10.15	11.29	8.76
17	30 × 4 × 4	5.73	5.73	5.53	12.44	13.90	15.19	10.33	7.58	5.70	12.10	11.51	13.76
18	30 × 4 × 6	7.55	7.55	6.09	15.69	17.82	18.21	8.97	9.08	8.05	13.52	12.57	12.85
19	40 × 2 × 2	0.76	0.10	0.00	6.20	6.72	8.57	3.30	0.28	0.07	5.07	5.10	5.21
20	40 × 2 × 4	1.18	0.43	0.30	7.97	8.57	10.81	2.69	0.96	0.33	7.16	6.79	8.33
21	40 × 2 × 6	2.09	0.98	0.98	9.48	13.66	12.40	2.94	1.79	1.16	10.57	10.32	11.15
22	40 × 3 × 2	5.19	0.05	0.05	17.50	17.09	15.71	6.41	1.18	0.14	13.93	12.44	11.15
23	40 × 3 × 4	5.21	1.51	0.34	19.93	21.81	23.20	6.12	2.46	1.28	14.80	13.88	15.82
24	40 × 3 × 6	2.40	2.40	2.40	22.89	22.84	25.71	5.72	5.86	2.86	20.44	16.36	26.64
25	40 × 4 × 2	6.42	0.74	0.65	22.65	29.14	25.43	12.63	2.32	1.05	25.48	29.47	18.94
26	40 × 4 × 4	2.82	1.35	1.02	24.70	30.45	31.11	9.90	5.20	2.97	23.86	33.02	30.00
27	40 × 4 × 6	3.89	3.43	3.43	29.63	33.51	31.39	9.06	7.06	6.49	35.04	28.37	26.85
28	50 × 2 × 2	0.81	0.00	0.00	7.34	11.63	11.94	3.39	0.59	0.12	9.96	9.03	7.97
29	50 × 2 × 4	1.65	0.52	0.20	9.11	15.16	12.94	3.73	1.01	1.23	12.56	13.57	13.00
30	50 × 2 × 6	2.04	1.32	1.08	10.07	14.19	14.80	3.71	1.89	1.39	13.58	12.62	19.73
31	50 × 3 × 2	1.84	0.03	0.03	22.66	31.38	22.49	8.30	1.78	0.88	25.04	21.02	12.55
32	50 × 3 × 4	2.08	0.31	0.03	26.03	37.71	32.29	6.93	2.27	1.40	28.71	31.48	25.64
33	50 × 3 × 6	3.90	2.50	1.90	34.29	43.93	44.29	6.88	3.95	3.24	33.67	30.41	36.61
34	50 × 4 × 2	8.76	1.84	0.49	34.99	57.14	35.71	12.80	3.64	1.22	36.50	50.00	25.97
35	50 × 4 × 4	6.75	2.07	0.87	43.49	58.57	58.74	9.39	5.49	3.89	55.44	49.99	51.39
36	50 × 4 × 6	5.49	3.22	1.70	42.22	59.43	61.43	7.04	6.24	5.31	55.42	51.05	55.18
	Average	**3.07**	**1.56**	**1.24**				**6.97**	**3.78**	**2.94**			

Table 5.8 Comparison Results between WFA and MA on the Randomly Generated MA Instances

Test problem		WFA			MA		
No.	$N \times S \times m_l$	$\overline{\Delta}$	**SD (C_{max})**	***ratio***	$\overline{\Delta}$	**SDOFV**	***ratio***
1	10 × 2 × 2	0.00	0.00	1.00	0.00	0.00	1.00
2	10 × 3 × 2	5.00	0.00	8.91	11.76	0.00	18.92
3	10 × 4 × 2	21.35	0.00	18.36	39.20	0.60	206.25
4	10 × 5 × 2	25.30	0.00	29.55	44.38	0.90	254.17
5	10 × 6 × 2	33.33	0.00	40.27	55.53	1.30	504.17
6	20 × 2 × 2	0.00	0.00	56.36	13.78	0.56	102.08
7	20 × 3 × 2	6.67	0.16	303.64	14.29	0.94	631.25
8	20 × 4 × 2	16.99	0.09	535.45	20.16	1.40	958.33
9	20 × 5 × 2	16.53	0.39	812.73	29.85	1.30	1650.00
10	20 × 6 × 2	21.02	0.09	1269.09	45.33	2.10	1887.50
11	30 × 2 × 2	1.76	0.24	233.64	5.13	0.90	256.25
12	30 × 3 × 2	1.35	0.19	829.09	15.34	2.30	583.33
13	30 × 4 × 2	10.66	0.19	1426.36	30.83	1.96	702.08
14	30 × 5 × 2	18.80	0.21	2806.36	41.15	3.00	1477.08
15	30 × 6 × 2	20.71	0.75	6539.09	44.36	1.80	2997.92
16	40 × 2 × 2	0.42	0.00	613.64	0.95	0.00	270.83
17	40 × 3 × 2	5.22	0.50	1677.27	14.61	1.30	1250.00
18	40 × 4 × 2	10.83	0.69	3523.64	34.07	2.20	2777.08
19	40 × 5 × 2	14.43	0.69	6541.82	34.40	2.20	2908.33
20	40 × 6 × 2	12.80	1.19	9286.36	40.73	2.50	6995.83
	Average	**12.16**	**0.27**		**26.79**	**1.36**	

The results of comparison between the WFA and the tabu search algorithms of Wardono and Fathi (2004) on the modified Wittrock benchmark problems are shown in Tables 5.9, 5.10, and 5.11. We use the lower bound obtained by Wardono and Fathi (2004) for LB_{sol} in equation (5.5). From these tables, it can be seen that the WFA outperforms TS-H1 and TS-Z3 both in solution quality and CPU time ratio for most benchmark instances. Thus, the WFA has obtained improved solutions than that reported in Wardono and Fathi (2004), except for instance 1 in Table 5.11. For the FFSP with no available buffer space, the average relative percentage increase in objective value of the WFA is 0.95% less than that of TS-H1 (see Table 5.9). For the FFSP with finite buffers,

the average relative percentage increase of the WFA is 0.94% less than that of TS-H1 (see Table 5.10). For the FFSP with unlimited buffers, the average relative percentage increase of the WFA is 0.81% less than that of TS-Z3 (see Table 5.11). When solving instances with different sizes, it is also observed that the CPU time ratio of the WFA is less than that of TS-H1 and TS-Z3 for most instances.

Table 5.9 Comparison Results between WFA and TS-H1 for the FFSP with No Available Buffer Space

Problem	Instance	*LB*	WFA			TS-H1		
			C_{max}	$\bar{\Delta}_{Best}$	*ratio*	C_{max}	$\bar{\Delta}_{Best}$	*ratio*
Modified Wittrock	1	746.3	811	8.67	5.73	822	10.14	6.47
	2	758.0	826	8.97	2.89	839	10.69	3.29
	3	758.7	815	7.42	2.25	822	8.35	2.29
	4	755.3	820	8.57	1.46	825	9.22	4.30
	5	961.5	971	0.99	2.51	974	1.30	2.69
	6	666.7	682	2.29	1.00	686	2.90	1.00
Average				**6.15**			**7.10**	

Table 5.10 Comparison Results between WFA and TS-H1 for the FFSP with Finite Buffer Capacities

Problem	Instance	*LB*	WFA			TS-H1		
			C_{max}	$\bar{\Delta}_{Best}$	*ratio*	C_{max}	$\bar{\Delta}_{Best}$	*ratio*
Modified Wittrock	1	746.3	763	2.24	6.78	776	3.98	8.83
	2	758.0	767	1.19	2.82	774	2.11	3.26
	3	758.7	770	1.49	2.25	777	2.42	2.37
	4	755.3	772	2.21	1.62	775	2.60	1.80
	5	961.5	962	0.05	1.45	969	0.78	1.40
	6	666.7	669	0.34	1.00	675	1.25	1.00
Average				**1.25**			**2.19**	

Table 5.11 Comparison Results between WFA and TS-Z3 for the FFSP with Unlimited Buffers

Problem	Instance	*LB*	WFA			TS-Z3		
			C_{max}	$\bar{\Delta}_{Best}$	*ratio*	C_{max}	$\bar{\Delta}_{Best}$	*ratio*
Modified Wittrock	1	746.3	760	1.83	6.70	760	1.83	6.99
	2	758.0	764	0.79	2.88	773	1.98	2.82
	3	758.7	770	1.49	2.05	776	2.28	1.86
	4	755.3	769	1.81	1.98	776	2.74	2.40
	5	961.5	962	0.05	1.43	969	0.78	1.45
	6	666.7	669	0.34	1.00	677	1.55	1.00
Average				**1.05**			**1.86**	

In addition, we also compared the results obtained by the WFA for the maltose syrup production problem to the optimal values obtained by using enumeration. The results are shown in Table 5.12 with two different *LB* values being used for the WFA. The first *LB* is based on using the optimal objective value, while the second *LB* is based on using the lower bounds derived from Wardono and Fathi (2004), Azizoglu et al. (2001) and Akturk and Yildirim (1998) respectively for the three different objective functions. As for LB_{sol} in equation (5.5), we only use the optimal objective value. Note that the objective values obtained by both the WFA and the enumeration method in Table 5.12 correspond to three different solutions according to the three respective objective functions. The results in this table show that the WFA is able to obtain the optimal solutions regardless of the two *LB* values used. Moreover, the WFA obtained these optimal solutions with significantly smaller computation time than that of the enumeration method.

From the experimental results for the benchmark instances, the randomly generated instances, and the instance from maltose syrup production, we conclude that the WFA is an efficient meta-heuristic algorithm for solving the FFSP with limited as well as unlimited buffers.

Table 5.12 Computational Results of WFA for Maltose Syrup Production Problem

Objectives used	Optimal solution obtained by enumeration		Solution obtained by WFA when optimal value is used for *LB*		Solution obtained by WFA when lower bound from literature is used for *LB*		
	Optimal value	Time (s)	$\overline{\Delta}$	Time (s)	*LB*	$\overline{\Delta}$	Time (s)
Makespan	139.00	130	0.00	1.6	109.00[a]	0.00	1.8
Total weighted flow time	75.95	106	0.00	1.8	63.10[b]	0.00	2.1
Total weighted tardiness	0.10	105	0.00	1.9	0.00[c]	0.00	1.9

[a] Lower bound derived from Wardono and Fathi (2004).

[b] Lower bound derived from Azizoglu et al. (2001).

[c] Lower bound derived from Akturk and Yildirim (1998).

5.6 Conclusions

In this chapter, we propose a WFA for solving the FFSP. It involves using an erosion capability relationship function between the amount of precipitation and its falling force to create a flexible operation scheme for the erosion process. This helps the erosion process to focus on exploiting promising regions strongly. We show how this algorithm can be applied to solve the FFSP with limited as well as unlimited buffers. In addition, we also propose an improved procedure from that of Wardono and Fathi (2004) for constructing a complete schedule of the FFSP problem. Computational experiments and comparisons were carried out to show the performance of the proposed algorithm. The results show that the WFA is a promising algorithm not only for solving benchmark instances and randomly generated instances but also for solving problems arising in practical applications. Improved solutions to benchmark problems are also found by our proposed algorithm. Some preliminary results were first reported in Tran and Ng (2009), and the full results of this chapter were then reported in Tran and Ng (2010).

CHAPTER 6

MOWFA FOR MULTI-OBJECTIVE SCHEDULING

In this chapter, we construct a multi-objective water flow algorithm (MOWFA) for solving multi-objective scheduling problems. In particular, we investigate the multi-objective flexible flow shop scheduling problem (MOFFSP) with limited intermediate buffers. Two objectives of this scheduling problem are the minimization of the completion time of jobs and the minimization of the total tardiness time of jobs. In the MOWFA, landscape analysis is performed to determine the weights of objective functions, which guide the drops of water to exploit potential regions and move towards the optimal Pareto optimal solution set. We also include the evaporation and precipitation processes in this algorithm to enhance the solution exploitation capability of the algorithm in potential neighboring regions. In addition, we propose an improvement process for reinforcing the final Pareto solution set obtained. The performance of the MOWFA is tested with benchmark instances taken from the literature and randomly generated instances. The computational results and comparisons demonstrate the effectiveness and efficiency of the proposed algorithm.

Chapter 6 is organized as follows. In Section 6.1, we introduce the MOFFSP with limited intermediate buffers and describe its applications. A brief literature review of

research works on the multi-objective scheduling problem is also presented in this section. Section 6.2 describes the details of the MOFFSP with limited intermediate buffers. Then, the proposed MOWFA for solving the multi-objective scheduling problem is discussed in detail in Section 6.3. Computational results and comparisons based on the benchmark instances taken from the literature and randomly generated instances are shown in Section 6.4. Finally, some conclusions of this chapter are presented in Section 6.5.

6.1 Introduction

Flexible flow shop scheduling (FFSP) with limited buffers is one of the well-known scheduling problems due to its important applications in both traditional and modern manufacturing systems. A brief description of the practical applications of this scheduling problem can be found in Section 5.1. The FFSP with limited buffers is known to be an NP-hard problem (Wardono and Fathi, 2004). It becomes even more complex when we need to solve the problem in real-life production environment where several conflicting objectives are simultaneously considered. Two primary objectives of the problem investigated in this chapter are minimization of the completion time of jobs (makespan or C_{max}) and minimization of the total tardiness time of jobs.

Quadt and Kuhn (2007) and Ribas et al. (2010) presented a detailed review of single-objective scheduling algorithms and the classification of FFSP. Although there have been a lot of research works on the FFSP, only a few of them dealt with the FFSP with limited buffers. There are also limited research works on the multi-objective scheduling problem. An extensive review of multi-objective scheduling problems in the past 13 years is presented by Lei (2009). From this review, only Wei et al. (2006)

proposed an evolutionary algorithm for solving the MOFFSP. However, the authors did not consider the FFSP with limited buffers. Qian et al. (2009) presented a hybrid differential evolution algorithm for solving the multi-objective permutation flow shop scheduling problem with limited buffers, in which there is one machine at each stage.

Recently, Rashidi et al. (2010) proposed a hybrid parallel genetic algorithm for the MOFFSP. In their paper, the authors investigated the FFSP with unrelated parallel machines, sequence-dependent setup times, and processor blocking to minimize the makespan and the maximum tardiness. The proposed solution procedure consists of independent parallel genetic algorithms in which each genetic algorithm searches for optimal solutions in different directions based on different assigned weights for each subpopulation. The computational results show that it is an efficient algorithm for the MOFFSP that was considered. However, assigning different weights to subpopulations may not be efficient as one individual chromosome could be assigned with more than a pair of weights. Moreover, the algorithm is dependent on a large parameter set defined by the user. In addition, the solution representation is based on random keys, which increases the computation time due to the need for decoding.

In this chapter, we propose the MOWFA for solving the MOFFSP with limited intermediate buffers. This algorithm integrates several search procedures for solving the multi-objective scheduling problem efficiently. In particular, in the exploration phase of the MOWFA, the FFSP is divided into many scheduling sub-problems. Drops of water (*DOW*s) in the sub-problems are assigned suitable weights based on landscape analysis to search for optimal solutions in the corresponding directions. In the exploitation phase of the MOWFA, the erosion process with local and global neighborhood structures guides *DOW*s to overcome obstacles to search for better

optimal solutions in the region in which they are generated, and in the neighboring regions, respectively. Also, we include an evaporation process in the algorithm to enhance the solution exploitation capability of this algorithm in potential neighboring regions. In addition, we propose an improvement search process for reinforcing the final Pareto solution set obtained. The Wittrock benchmark instances taken from the literature, as well as randomly generated instances, are used to evaluate the performance of the proposed algorithm. The computational results and comparisons show that the MOWFA is an efficient nature-inspired algorithm for solving the MOFFSP with limited intermediate buffers.

6.2 MOFFSP with Intermediate Buffers

A detailed description of the FFSP with intermediate buffers has been provided in Section 5.2. In this chapter, we only consider two cases of the FFSP with intermediate buffers. The first case is the FFSP with no buffer between consecutive stages. Thus, if there is no idle machine in the subsequent stage $l + 1$, a job completed on a machine in stage l must then wait until at least a machine in stage $l + 1$ is available. The second case is the FFSP with finite buffer capacities between consecutive stages. In this case, if there is no available machine in the subsequent stage $l + 1$, then a job completed on a machine in the previous stage l may wait in a following buffer l. If there is no available buffer space, it remains on the blocked machine in stage l until there is an available buffer space or an idle machine in stage $l + 1$.

Since intermediate buffers can be considered as machines with zero processing time, the FFSP with finite buffers can be converted to one with no available buffer space (McCormick et al., 1989). As a result, we only need to construct an MOWFA for

solving the MOFFSP with no buffer. Then, we can apply the MOWFA for solving the MOFFSP with finite buffers by a modification of the input data described in Section 5.2. In the transformed problem, a buffer at stage l with a storage capacity B_l is considered as a stage with B_l identical parallel machines (see Figure 6.1). The processing time of jobs on the machines in the buffer stages is zero. Then, the total number of stages in this problem becomes $2S - 1$. Every job must be processed through all stages, including the buffer stages.

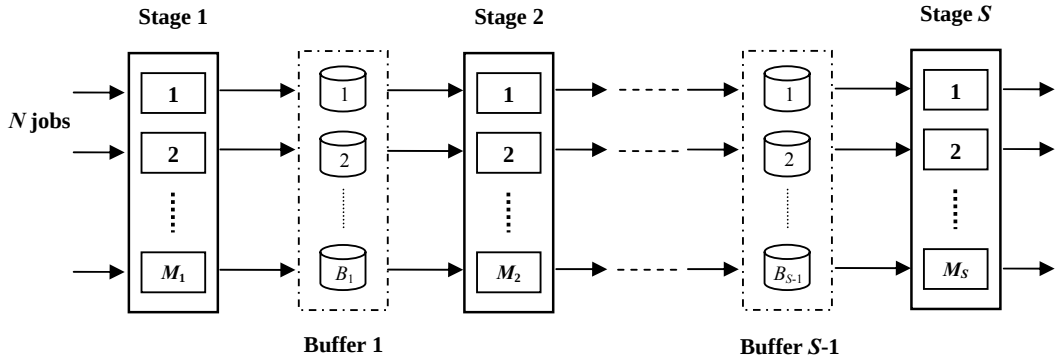


Figure 6.1 FFSP with Operation Stages Including Intermediate Buffers

Let π denote a job permutation, C_{jl} denote the completion time of job j at stage l, d_j denote the deadline of job j, and T_j denote the tardiness time of job j. The tardiness time of job j can be defined as $T_j = \max\{C_{jS} - d_j, 0\}$. Then, two objective functions used in the multi-objective scheduling problem, namely minimizing the completion time of jobs $f_1(\pi)$ and minimizing the total tardiness time of jobs $f_2(\pi)$, can be determined by:

$$\min f_1(\pi) = \max_{j \in N} \{C_{jS}\}, \tag{6.1}$$

$$\min f_2(\pi) = \sum_{j=1}^{N} T_j. \tag{6.2}$$

These objectives help to achieve a high throughput for production.

The general MOFFSP with the two objectives mentioned earlier can be described as follows:

$$\min \quad f(\pi) = [f_1(\pi), f_2(\pi)], \tag{6.3}$$

$$\text{subject to} \quad \pi \in \Pi, \tag{6.4}$$

where Π denotes the set of possible job permutations.

To solve the multi-objective optimization problem, we use the concepts of Pareto optimality described below in terms of a minimization problem:

(a). ***Pareto dominance***: a job permutation π_1 dominates another job permutation π_2, denoted as $\pi_1 \succ \pi_2$, if and only if we have:

$$f_h(\pi_1) \le f_h(\pi_2), \ \forall h \in \{1,2\} \text{ and } \exists k \in \{1,2\}, \ f_k(\pi_1) < f_k(\pi_2).$$

(b). ***Pareto optimal job permutation***: a job permutation π_1 is considered to be a Pareto optimal job permutation if and only if there is no job permutation $\pi_2 \in \Pi$ that dominates π_1.

(c). ***Pareto optimal set***: Pareto optimal set is a set of all Pareto optimal job permutations.

(d). ***Pareto optimal front***: Pareto optimal front is the set of all objective values corresponding to the job permutations in the Pareto optimal set.

(e). ***Non-dominated job permutation***: a job permutation π is said to be non-dominated with respect to a given set of job permutations if and only if π is not dominated by any job permutation in the given set.

In the Pareto optimal set, the job permutations cannot be improved in any objective function without degrading the value in at least one other objective function. Hence, there is no job permutation that is the best for all objectives. As a result, multi-objective optimization algorithms often need to find a set of good non-dominated job permutations. To compare the performance of multi-objective optimization algorithms, three main aspects are considered: the number of non-dominated job permutations in the Pareto set obtained, the distance of the Pareto front obtained to the Pareto optimal front, and the spread of non-dominated job permutations in the Pareto set obtained (Silva et al., 2004).

In addition, we use a single-objective optimization approach to solve the multi-objective scheduling problem. In this approach, the multi-objective optimization problem is transformed to a single-objective optimization problem by combining objective functions and their weights linearly. The main drawback of this approach is that the weights of the objective functions are subjectively provided. To overcome this drawback, we have performed landscape analysis of the multi-objective optimization problem to determine correlation among objective functions. Then, weights are automatically updated and assigned to corresponding objective functions. The single-objective optimization approach is used to search for local optimal job permutations of the multi-objective problem in the MOWFA. The details of performing the procedure will be described in the next section.

6.3 MOWFA for the MOFFSP with Intermediate Buffers

In this section, we describe the procedure of MOWFA for solving the MOFFSP with intermediate buffers, including the solution representation, procedure for constructing

a complete schedule, erosion condition, erosion capability, and erosion process. Other important operators of the MOWFA for this problem, such as landscape analysis, evaporation process, and improvement process, are also described.

6.3.1 Encoding Scheme

In the MOFFSP with intermediate buffers, a *DOW* is associated with a job permutation. The job permutation provides the longitude and latitude information for the position of *DOW* on the ground. Its corresponding objective values provide the altitude information for the position of the *DOW*. Given a job permutation $\pi = (\delta_1, ..., \delta_N)$, we define:

$$\text{longitude}(\pi) = (\delta_1, ..., \delta_{\lfloor \frac{N}{2} \rfloor}), \quad \text{and} \quad \text{latitude}(\pi) = (\delta_{\lfloor \frac{N}{2} \rfloor + 1}, ..., \delta_N), \tag{6.5}$$

where $\lfloor q \rfloor$ is the largest integer less than or equal to q. We also define the altitude of the *DOW* as comprising of the two objective functions, makespan and total tardiness time of the job permutation π. In Figure 6.2, we illustrate an example of a *DOW* and its positional vector components for the MOFFSP with 8 jobs.

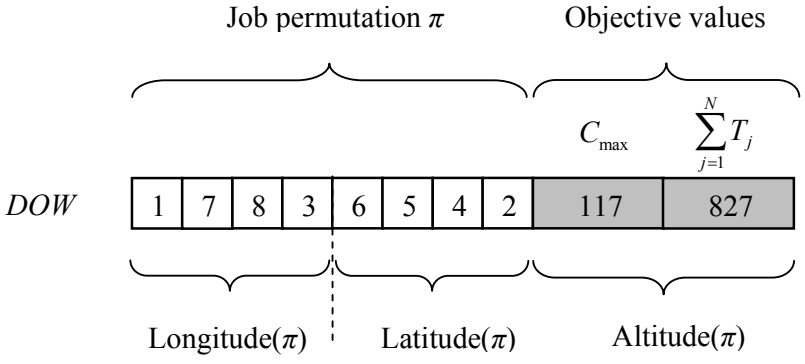


Figure 6.2 An Example of Solution Representation in MOWFA for the MOFFSP

In the MOWFA, the job permutations providing the longitude and latitude information represent the sequence of N jobs processed in the first stage. However, the job permutation cannot fully determine a complete schedule for the jobs going through all the stages, and thus the corresponding objective values are also not determined. As such, we use the H1-variant procedure described in Section 5.3.1 to construct a complete schedule associated with a given job permutation. This procedure has the advantage that it is applicable to the FFSP with no buffer and the FFSP with unlimited buffers. Also, with this procedure, the search space of the problem is limited to a set of possible permutations of N jobs. Then, we do not need to monitor the number of machines in each stage, and the computational procedure of the MOWFA becomes simplified. As in the case of the single-objective FFSP in Chapter 5, although using the job permutation representation for the multi-objective scheduling problem may not include the global optimal job permutation of this problem, the best job permutation found by the WFA or any algorithm with this representation is very near the global optimal job permutation. Moreover, the job permutation representation is more easily integrated into metaheuristic algorithms than the matrix representation, which can cover the entire solution space but requires computation time to determine whether a solution given in this representation is feasible (Wardono and Fathi, 2004).

6.3.2 Memory Lists

In the MOWFA, we use three memory lists in the search for the Pareto optimal set. The first list, denoted as the Pareto-list, stores the best non-dominated job permutations. The second list, denoted as the UE-list, is used to store local optimal job permutations which have not been eroded due to the erosion condition not being satisfied. The list aims to record the potential job permutations to be considered for

performing the erosion process. The final list, denoted as the E-list, is used to store local optimal job permutations eroded. The E-list aims to prevent regenerating *DOW*s into the job permutations that are eroded in the next iterations and would thus help to save the computational time of the algorithm.

In our approach, multiple objective functions are linearly combined into a single objective function so that the search for the local optimal job permutations is based on a scalar objective function. Updating the UE-list and the E-list is also based on the local optimal job permutations with respect to the scalar objective function. However, updating the Pareto-list is based on the property of *Pareto dominance* in multi-objective optimization.

6.3.3 Exploration Phase

Here, we describe the operational mechanism of the components in the exploration phase of MOWFA. This phase aims to spread the *DOW*s to many places on the ground to increase the solution diversification capability of the algorithm. It also helps the *DOW*s to select suitable directions to search for the Pareto optimal set by analyzing the landscape, which represents a set of objective values of feasible solutions.

6.3.3.1 Distinct Regions

In the exploration phase of this algorithm, we divide the MOFFSP with intermediate buffers into multiple scheduling sub-problems so that the solution space of the problem is divided into N distinct regions, where N is the instance size. This is achieved by fixing the first position of a job in a job permutation from 1 to N when generating the

job permutations of *DOW*s. The rest of the positions in a job permutation are assigned randomly. This can be represented with the following notation:

$$\Omega = \{\Omega_1, \Omega_2, ..., \Omega_N\}, \text{ with } \Omega_i = \{\ \pi\ |\ \pi[1] = i\ \} \text{ for } i = 1, 2, .., N, \tag{6.6}$$

where Ω denotes the solution space of MOFFSP, and Ω_i denotes i^{th} distinct region.

6.3.3.2 Landscape Analysis

Landscape analysis for the distinct regions is performed to determine suitable searching directions for the *DOW*s generated in the regions. Here, the searching direction is defined by a pair of weights (c_1, c_2) that is used to combine the multiple objective functions into a single objective function. Thus, the scalar objective function $f(\pi)$ is defined as:

$$f(\pi) = c_1 f_1(\pi) + c_2 f_2(\pi), \tag{6.7}$$

where c_1 and c_2 are the weights of objective function $f_1(\pi)$ and $f_2(\pi)$, respectively. Based on the pair of weights, *DOW*s will search for local optimal job permutations in the corresponding direction.

In this chapter, we use the landscape analysis approach proposed by Tantar et al. (2008). At an iteration, we randomly generate a sample of *MaxPop* job permutations for each distinct region Ω_i. An approximation of the enclosing ellipse for the feasible solution space in the distinct region is then obtained by such a sampling as described in Tantar et al. (2008). Here, a complete ellipse is defined by the center coordinates (x_0, y_0), the orientation angle α, and the major axis a and minor axis b. The orientation angle provides the correlation degree that exists between the objective functions (see Figure 6.3). Hence, finding non-dominated job permutations with an appropriate

orientation angle will lead to the Pareto optimal set. Given the orientation angle α, a pair of corresponding weights (c_1, c_2) is determined by:

$$c_1 = \cos^2(\alpha), \tag{6.8}$$

$$c_2 = \sin^2(\alpha). \tag{6.9}$$

In the subsequent iterations, the approximation of the enclosing ellipse for region i is determined based on the cumulative sample set of job permutations generated in the region. When the sample size increases up to 100, we use the pair of weights (c_1, c_2) obtained at the iteration for the corresponding region in the remaining iterations. We do not need to recalculate the enclosing ellipse since the difference between the 100-sample set and any larger sample set is almost unnoticeable (Tantar et al., 2008). This will help to reduce computation time for performing the MOWFA.

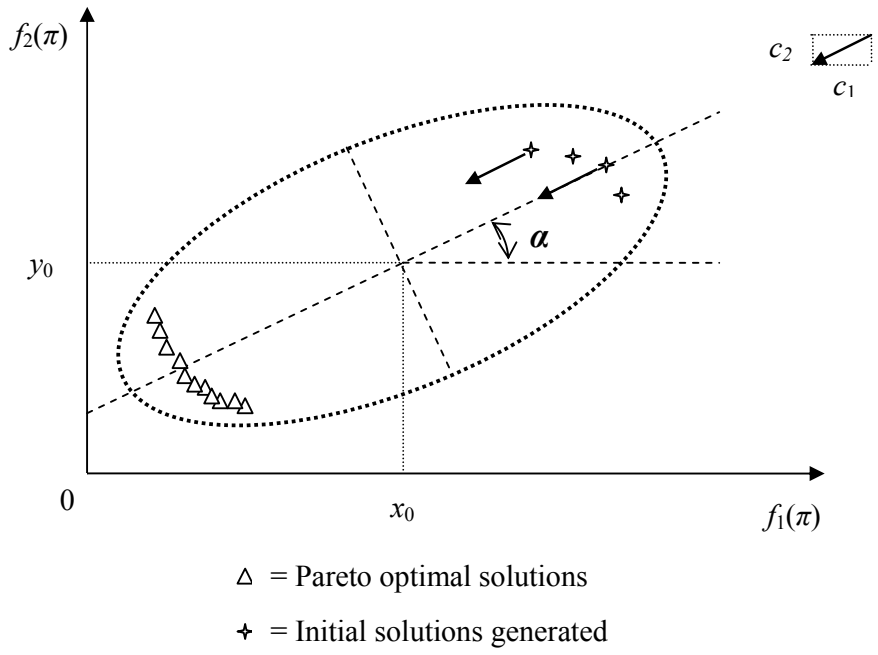


Figure 6.3 An Illustration for Finding the Pareto Set Based on the Orientation Angle in MOWFA

6.3.3.3 Seed Job Permutations

To improve the performance of the MOWFA, we generate a seed job permutation in the initial iteration by using the flexible flow line loading (FFLL) algorithm (Pinedo, 2005). A brief description of this algorithm is provided in Section 5.3.3. In order to generate a set of N job permutations corresponding to N distinct regions, we use a swap scheme based on the seed job permutation. This scheme is performed by interchanging only the job at the first position with a job at a different position in the seed job permutation. The FFLL algorithm is only used in the initial iteration. For subsequent iterations, the FFLL algorithm is no longer used and instead, we use randomly generated job permutations for each distinct region.

6.3.3.4 Hill-Sliding Algorithm

For the sample set of job permutations generated, including the seed job permutations obtained by the swap scheme, we apply the single-objective optimization approach to search for local optimal job permutations. Here, the steepest descent hill-sliding algorithm described in Section 4.3.3 is used to guide *DOW*s to reach the local optimal positions corresponding to the scalar objective function. Note that the pair of weights (c_1, c_2), determined by approximating an enclosing ellipse for feasible solution space in the corresponding distinct region, is used for transforming the multi-objective FFSP into a single-objective FFSP.

These local optimal job permutations and the number of *DOW*s at the job permutations are updated in the UE-list to be considered for performing the erosion process in the subsequent exploitation phase.

6.3.4 Neighborhood Structures

In this chapter, the neighborhood structures used consist of the swap scheme described earlier and the 2-opt neighborhood structure. The 2-opt neighborhood structure determines all neighboring job permutations that can be obtained from a current job permutation by exchanging positions of two jobs in the current job permutation. For example, if π' denotes the job permutation obtained by exchanging the positions of two jobs δ_i and δ_j in a job permutation π, then π' can be expressed as:

$$\begin{aligned} &\pi'[\delta_i] = \pi[\delta_j],\ \pi'[\delta_j] = \pi[\delta_i], \\ &\pi'[\delta_k] = \pi[\delta_k] \ \text{ for } \ k \in N \setminus \{i, j\}, \end{aligned} \tag{6.10}$$

where $\pi[\delta]$ and $\pi'[\delta]$ denote the position of job δ in the job permutation π and its neighboring job permutation π', respectively.

In the MOWFA, we divide the 2-opt neighborhood structure into two types, namely, local and global 2-opt neighborhood. In the local 2-opt neighborhood, we do not exchange the job in the first position with a job in any other positions in a job permutation. The neighborhood structure is used for finding local optimal job permutations in distinct regions at the exploration phase. It is also used in the erosion process. The neighborhood structure aims to find the Pareto solution set in a specific distinct region. With the local 2-opt neighborhood, *DOW*s only move in the distinct region in which they are generated. The number of neighboring job permutations obtained by the neighborhood structure is $(N–1)(N–2)/2$. Otherwise for the global 2-opt neighborhood, we can exchange the job in the first position with a job in any other position of a job permutation. Thus, the global 2-opt neighborhood aims to guide the *DOW*s to explore and exploit potential neighboring regions (see Figure 6.4). The

number of neighboring job permutations obtained by the neighborhood structure is $N(N-1)/2$. It is used in the erosion process and the improvement phase.

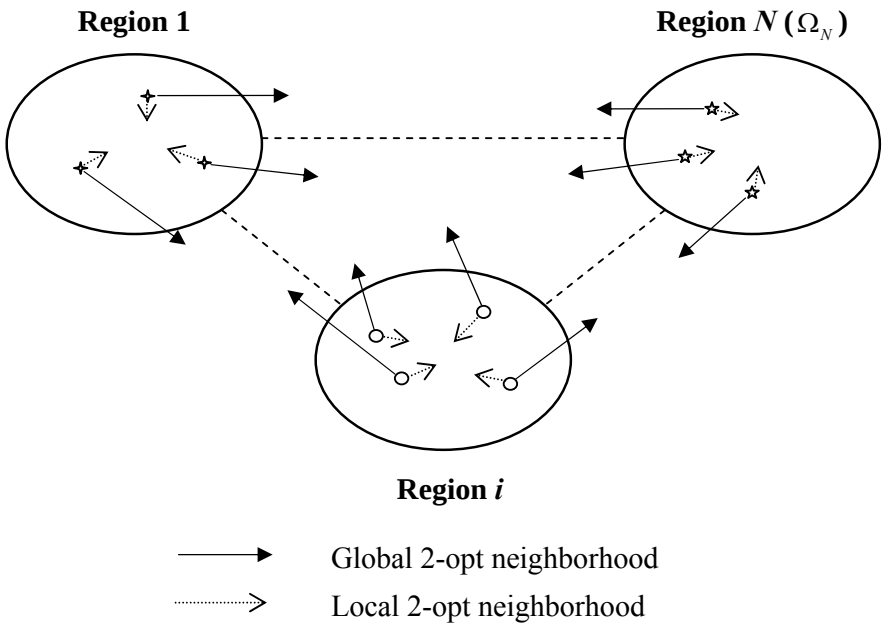


Figure 6.4 An Illustration of the Search Direction of *DOW*s on Two Types of 2-opt Neighborhood

6.3.5 Exploitation Phase

In this section, we present the conditions to perform erosion process for the job permutations in the UE-list, the determination of erosion capability of *DOW*s at the job permutations satisfying the conditions, and two schemes of erosion process. The erosion process is the main operator of the exploitation phase which guides the *DOW*s to overcome the local optimal positions and obtain better or optimal positions.

6.3.5.1 Erosion Condition and Capability

In the MOWFA for the MOFFSP with intermediate buffers, the condition to perform erosion process is determined based on the amount of precipitation as described in

Section 4.3.4. Let *MinEro* denote the minimum amount of precipitation allowed to start the erosion process. Then, the erosion process will happen at a local optimal job permutation in the UE-list in which the amount of precipitation reaches *MinEro*. In the MOWFA, the amount of precipitation is represented by the number of *DOW*s at the eroding local optimal job permutation.

In the MOWFA, the erosion capability is based on two main factors, the amount of precipitation and its falling force. The amount of precipitation is represented by the number of *DOW*s as mentioned earlier, while its falling force depends on the altitude of positions of *DOW*s. The falling force of precipitation to lower positions is stronger than that of higher positions and would thus erode more easily for lower positions. Hence, the erosion capability becomes stronger for local optimal job permutations with larger number of *DOW*s and lower objective values.

Let *MaxUIE* denote the maximum number of iterations for the erosion process. We use the following relationship between the erosion capability and the above two factors as in Section 5.3.4.1:

$$MaxUIE = \varphi_1 Q(\pi^*) + \varphi_2^{\,LB/f(\pi^*)}, \tag{6.11}$$

where φ_1 and φ_2 represent the impact of precipitation and its falling force, respectively. Based on preliminary experimentation, we set the values for φ_1 and φ_2 to 2 and 3, respectively. Here, $Q(\pi^*)$ is the number of *DOW*s at the local optimal job permutation π^*, $f(\pi^*)$ is the scalar objective value of the local optimal job permutation, *LB* is the scalar lower bound that is defined as:

$$LB = c_1 LB_1 + c_2 LB_2, \tag{6.12}$$

where (c_1, c_2) is the corresponding pair of weights used to determine $f(\pi^*)$, and LB_1 and LB_2 represent the lower bounds of makespan and total tardiness time of jobs,

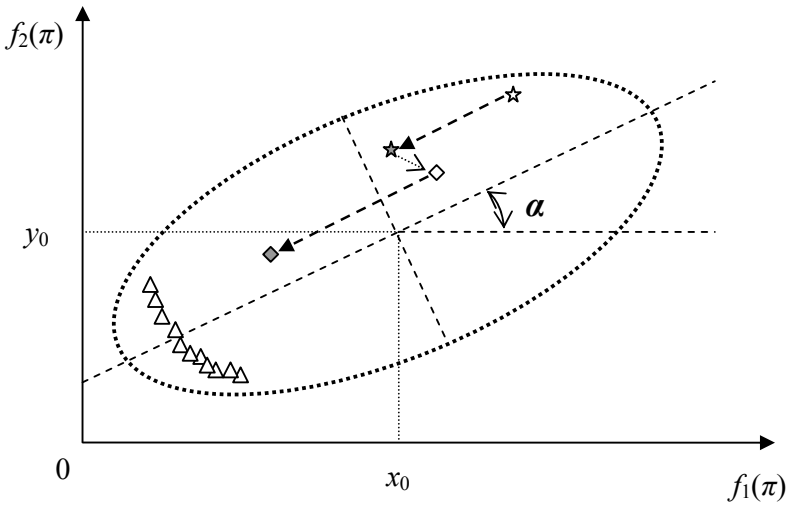
respectively. We have used the lower bound proposed by Wardono and Fathi (2004) for LB_1 and the lower bound proposed by Akturk and Yildirim (1998) for LB_2.

6.3.5.2 Erosion Process

In the MOWFA, when the erosion condition at some local optimal job permutation is satisfied, the erosion process will be carried out. Before performing the erosion process, the erosion capability of *DOW*s at the local optimal job permutation is determined using equation (6.11). Then, the erosion process is performed following an erosion strategy based on a topological parameter representing the geographical surface of the optimization problem. For the MOFFSP, the topological parameter Δd_h is defined as the difference between the scalar objective value of the local optimal job permutation π^* and that of its neighboring job permutation π_h^*:

$$\Delta d_h = f\left(\pi_h^*\right) - f\left(\pi^*\right). \tag{6.13}$$

In the MOWFA for the MOFFSP, we use the two neighborhood structures described in Section 6.3.4, i.e., the local and global 2-opt neighborhoods. With each neighborhood structure, we have a specific erosion scheme for finding better local optimal job permutations. The erosion scheme I, based on the local 2-opt neighborhood, only searches for better local optimal job permutations in a given distinct region, so that the *DOW*s only exploit a distinct region from which they are generated (see Figure 6.5). The erosion scheme II, based on the global 2-opt neighborhood, can search for better local optimal job permutations in other regions, so that the *DOW*s can move and exploit neighboring regions different from the region that they are generated (see Figure 6.6).



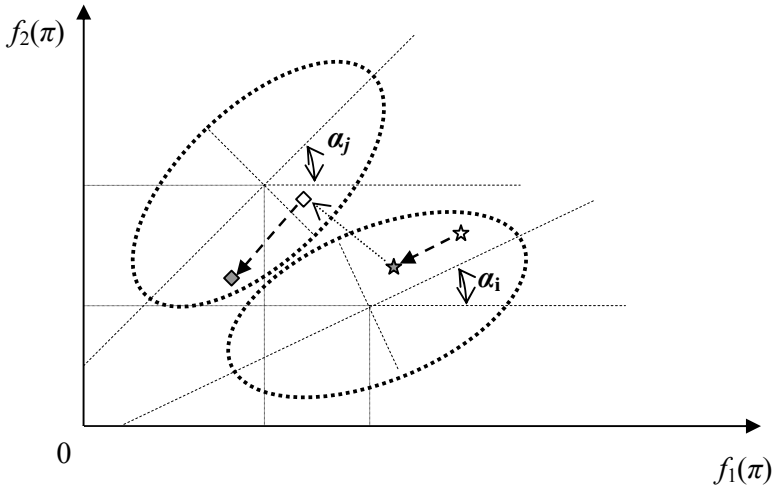
△ = Pareto optimal solutions

☆ = Initial solution generated

◆ ★ = Corresponding local optimal solutions

◇ = Intermediate solution with at least one worse objective value

Figure 6.5 An Illustration for Scheme I of the Erosion Process with the Local 2-Opt Neighborhood



☆ = Initial solution generated in region *i*

★ = Local optimal solution in region *i*

◇ = Solution found in region *j* after using the global 2-opt

◆ = Local optimal solution in region *j*

Figure 6.6 An Illustration for Scheme II of the Erosion Process with the Global 2-Opt Neighborhood

In this erosion process, the E-list will also prevent the *DOW*s from moving to eroded job permutations in other regions. Hence, the *DOW*s will only move to un-eroded regions where their scalar objective value is improved.

6.3.6 Evaporation and Precipitation

In the MOWFA, we include an evaporation process in the algorithm to remove poor local optimal job permutations in the UE-list. A precipitation process is also used to regenerate the evaporated *DOW*s into neighboring regions, which reinforces the erosion possibility in the regions.

In the evaporation process, the local optimal job permutations with one *DOW*, held in the UE-list for a pre-specified number of iterations T, will be deleted from the list. Then, in the next iteration, a precipitation process will generate the number of deleted *DOW*s into new positions in neighboring regions. For example, if we remove one *DOW* in region i from the UE-list, we will randomly generate *DOW*s in regions $i - 1$ and $i + 1$. In the case of the *DOW* being deleted from region 1 or N, it will be randomly generated in regions N and 2, or regions $N - 1$ and 1, respectively.

6.3.7 Improvement Phase

The two schemes in the erosion process can be applied to a scalar objective function and they aim to exploit strongly promising regions as well as potential neighboring regions. However, since such an approach has been restricted to a scalar objective function, its performance may be limited. Hence, we propose an improvement process for the Pareto set obtained. This process is still based on the global 2-opt neighborhood. However, it is performed by directly comparing two job permutations

based on the original objective functions in a similar manner as Ravindran et al. (2005), which used the difference between the objective values obtained. To illustrate, suppose we have two job permutations π and π'. The makespan and total tardiness time of jobs for these two job permutations are denoted by C_{max}, ST, and C'_{max}, ST′, respectively. Then, we define the following comparison values R and R′ to determine which job permutation is better:

$$R = [(C_{max}–\text{Min}(C_{max},C'_{max}))/\text{Min}(C_{max},C'_{max})]+[(ST–\text{Min}(ST,ST'))/\text{Min}(ST,ST')] \quad (6.14)$$

$$R'= [(C'_{max}–\text{Min}(C_{max},C'_{max}))/\text{Min}(C_{max},C'_{max})]+[(ST'–\text{Min}(ST,ST'))/\text{Min}(ST,ST')] \quad (6.15)$$

If R is less then R′, then the best job permutation is π. Otherwise, the best job permutation is π'.

In summary, after the maximum number of allowed iterations (*MaxCloud*) has been reached, the exploration and exploitation phases terminate. Then, the final Pareto set obtained will be improved by the improvement process described above. The improvement process will terminate after $\lfloor N/3 \rfloor$ iterations.

A flow chart of the MOWFA for the MOFFSP with intermediate buffers is shown in Figure 6.7.

6.4 Computational Experiments and Comparisons

6.4.1 Generation of Test Problems and Benchmark Problem Set

Computational experiments have been conducted to evaluate the performance of the proposed MOWFA. The data required for the MOFFSP with intermediate buffers

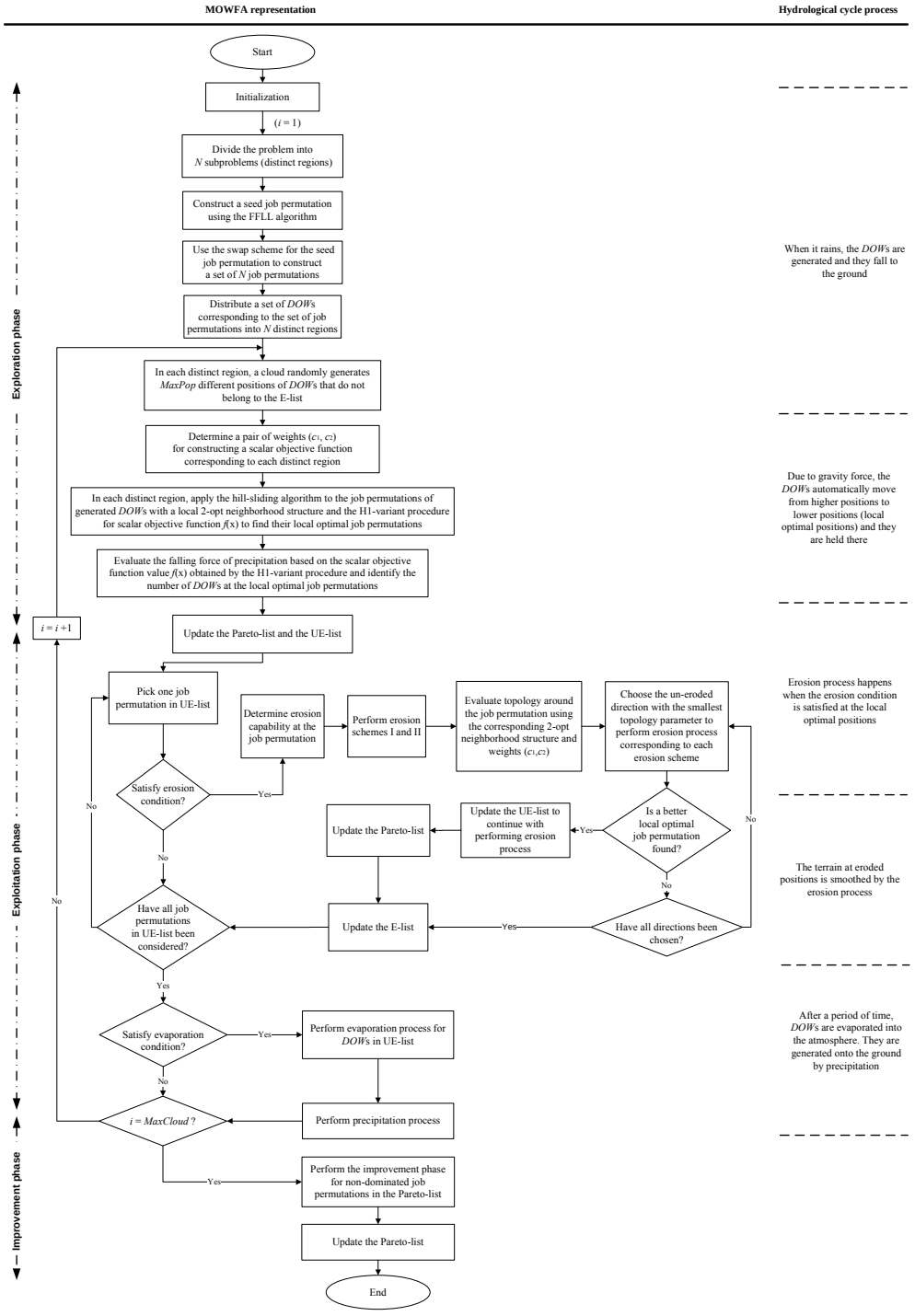


Figure 6.7 Flow Chart of the MOWFA for the MOFFSP

consist of the number of jobs, the number of stages, the number of machines at each stage, the number of buffers between consecutive stages, the processing times of jobs, and the due date of jobs. We generated the data of the test problems based on the procedure of generating datasets in Rashidi et al. (2010). We constructed the instances with the number of jobs N = 20, 40, 60, 80, 100; the number of stages S = 2, 4; and the number of machines in each stage is uniformly distributed in the interval [1, 4]. For each job, its processing time is uniformly distributed in the interval [10, 100]. When solving the instances with finite buffers, the buffer capacity for all stages in all instances is set to be 3.The due date of a job is set as follows:

$$d_j = \max_{j \in N} \left\{ \sum_{l=1}^{S} p_{jl} \right\} + \frac{LB_1}{10} \times U, \tag{6.16}$$

where p_{jl} denotes the processing time of job j in stage l, and LB_1 denotes the lower bound of makespan obtained by Wardono and Fathi (2004). Here, U denotes a uniform random number between 0 and 1. This results in a tight due date for the jobs. For each combination of jobs and stages, we have generated five different instances.

The benchmark problem set of Wittrock (1988), modified by Wardono and Fathi (2004), was also used to test the performance of the MOWFA. In these instances, the transport time (one minute) for the Wittrock instances is added to the processing time of jobs at stages 2 and 3, respectively, where applicable. The number of machines at stages 1, 2, and 3 are 2, 3, and 3, respectively. When solving the instances with finite buffers, we set the buffer capacity to be 3 for all stages.

In this chapter, we have compared the results obtained by the MOWFA with those obtained by the improved hybrid multi-objective parallel genetic algorithm (IHMOPGA) of Rashidi et al. (2010) for the generated test problems and the

benchmark instances. The IHMOPGA is designed for solving the FFSP with unrelated parallel machines, sequence-dependent setup times, and processor blocking, which is one specific scenario of the general FFSP with intermediate buffers that is considered in this chapter. Hence when compared with the MOWFA in the multi-objective problem, the procedure of IHMOPGA is still the same as in Rashidi et al. (2010). The only change in the IHMOPGA is in the input data. In particular, parallel machines at each stage are identical, and the setup time is zero for all jobs at all machines in each stage. Moreover, we also use the approach of converting the FFSP with finite buffers to one with no available buffer space as described in the MOWFA for the IHMOPGA, as the algorithm is used to solve the FFSP with finite buffers.

6.4.2 Platform and Parameters

All the computational experiments described in this chapter were performed on an Intel Centrino Duo 1.60 GHz CPU with 1.5 GB of RAM running on Windows XP Operating System. The MOWFA and IHMOPGA have been coded using Microsoft Visual Basic 6.0. Here, the computational complexity of the MOWFA for MOFFSP is determined based on the neighborhood structure used and the erosion process of this algorithm. In particular, the MOWFA used 2-opt neighborhood structure, and the worst possibility of the erosion process is to find for all n directions. Thus, the computational complexity of the MOWFA is estimated to be $O(n^3)$.

The choice of parameters for the MOWFA was determined by design-of-experiment methods. In particular, we carried out several simulations that test the MOWFA on instances with various values for the controlled parameters, *MaxCloud*, *MaxPop*, *MinEro*, and *T*, and we chose the best values for each instance. These best parameter

values are shown in Table 6.1. For the IHMOPGA, we used the same parameter values as Rashidi et al. (2010). In their paper, the authors did not mention exactly what method is used to choose their parameters.

When comparing with the IHMOPGA, we used the CPU time limit stopping criterion of $N \times \overline{M} \times S \times 25$ as presented in Rashidi et al. (2010). Here, $\overline{M}$ denotes the average number of machines in S stages. For the Wittrock instances, we ran the MOWFA and IHMOPFA with 5 independent replicates for each instance. The best results obtained from these replicates were used to compare the performance of MOWFA and IHMOPGA. For the randomly generated instances, we have used the average results of 5 different instances corresponding to a combination of jobs and stages to compare the performance of MOWFA and IHMOPGA. Rashidi et al. (2010) also used 5 different replicates for each experiment when they ran the IHMOPGA.

Table 6.1 Parameter Sets of the MOWFA for Instances of the MOFFSP

Instances	Parameter values			
	MaxCloud	*MaxPop*	*MinEro*	*T*
Modified Wittrock	10	5	3	3
Generated Dataset	10	10	3	3

6.4.3 Performance Metrics

In multi-objective optimization problems, proper comparison of algorithms is complex and is often based on the non-dominated solution set obtained by the algorithms. Three main aspects that are usually considered for evaluating the non-dominated solution set obtained by the algorithms are the number of non-dominated solutions in the Pareto set obtained, the distance of the Pareto front obtained to the Pareto optimal front, and the

diversity of non-dominated solutions in the Pareto set obtained (Silva et al., 2004). Comparison metrics can also be based on other aspects, such as a direct comparison between two algorithms based on the number of non-dominated solutions obtained by the algorithms or a comparison based on a union of non-dominated solutions obtained by the algorithms.

In this chapter, we use four comparison metrics representing all the above aspects. The first metric, denoted by Ψ_1, is the number of distinct non-dominated solutions obtained by each algorithm. The second metric, denoted by Ψ_2, is the fraction of the number of solutions in a non-dominated set obtained by an algorithm that are dominated by the non-dominated solutions obtained by another algorithm. The third metric, denoted by Ψ_3, is the percentage of non-dominated solutions obtained by each algorithm in the *Pareto optimal set*. The fourth metric, denoted by Ψ_4, is the distance of the non-dominated front obtained to the *Pareto optimal front*. As the *Pareto optimal set* and *Pareto optimal front* are usually not known in advance, we would choose distinct non-dominated solutions from a union of the non-dominated solutions obtained by all the compared algorithms to form the *Pareto optimal set* and *Pareto optimal front*, which are also the reference points in this comparison. This choice of the reference points may affect the reliability of the comparison results if the algorithm used to compare with the WFA has poor performance. Hence, to obtain reliable comparison results we only choose the IHMOPGA which has been evaluated to have good performance for solving the multi-objective FFSP.

Let P denote the union of non-dominated solutions obtained by the algorithms. Since we only compare the MOWFA with the IHMOPGA, P is thus a non-dominated solution set combined by P_1 and P_2, which denote the non-dominated solution sets

obtained by the MOWFA and the IHMOPGA, respectively. According to Van Veldhuizen (1999), the first metric is defined as $|P_1|$ for the MOWFA and $|P_2|$ for the IHMOPGA, respectively. Let D_1 and D_2 denote the number of solutions in the non-dominated solution set obtained by the MOWFA and the IHMOPGA that are dominated by the solutions in the IHMOPGA and the MOWFA, respectively. Then, the second metric is defined as:

$$\Psi_2^i = \frac{D_i}{P_i} \quad \text{for} \quad i = 1,2. \tag{6.17}$$

Let ND_1 and ND_2 denote the number of solutions in P_1 and P_2 which are not dominated by any other solutions in P, respectively. In particular, ND_1 and ND_2 are determined as follows:

$$ND_i = \left| P_i - \{\pi_i \in P_i \mid \exists \pi \in P : \pi \succ \pi_i\} \right| \quad \text{for} \quad i = 1,2. \tag{6.18}$$

Then, the third metric is defined as:

$$\Psi_3^i = \frac{ND_i}{P} \quad \text{for} \quad i = 1,2. \tag{6.19}$$

For the fourth metric, we use the following distance metric from Knowles and Corne (2002):

$$\Psi_4^i = \frac{1}{|P|} \sum_{\pi \in P} \min \left\{ d_{\pi_i,\pi} \mid \pi_i \in P_i \right\} \quad \text{for} \quad i = 1,2, \tag{6.20}$$

where $d_{\pi_i,\pi}$ denotes the distance between a solution π_i in the corresponding non-dominated solution set P_i and a reference solution π in the non-dominated solution set P. This is determined by:

$$d_{\pi_i,\pi} = \sqrt{(f_1^*(\pi) - f_1^*(\pi_i))^2 + (f_2^*(\pi) - f_2^*(\pi_i))^2} \quad \text{for} \quad i = 1,2, \tag{6.21}$$

where $f_i^*(\pi)$ denotes the *i*th objective normalized by using the reference solution set *P* in Ishibuchi et al. (2003).

Among the performance metrics, the first and third ones focus on the diversity of the non-dominated solution sets obtained, while the second and fourth ones focus on the closeness of the non-dominated set to the *Pareto optimal set*. Here, we want to maximize the diversity and minimize the closeness of the non-dominated solution set.

6.4.4 Computational Results

In this section, we provide the comparison results between the MOWFA and the IHMOPGA for the MOFFSP with intermediate buffers. Table 6.2 shows the results of MOWFA and IHMOPGA for the Wittrock instances with no available buffer capacity, while Table 6.3 shows the results for the instances with finite buffers.

From Tables 6.2 and 6.3, we see that the MOWFA outperforms IHMOPGA in all instances when we consider the metrics Ψ_2, Ψ_3, and Ψ_4. For metric Ψ_1, MOWFA is worse than IHMOPGA in instance 1 for the case of no buffer (Table 6.2), and in instances 1 and 6 for the case of finite buffer (Table 6.3). However, metric Ψ_1 is not significant when it is evaluated alone. It can be seen that at least 87.5% of the non-dominated solutions obtained by the IHMOPGA in instances 1 and 6 are dominated by the MOWFA (see the Ψ_2 metric column of IHMOPGA in Tables 6.2 and 6.3). Moreover, in instances 1 and 6, 100% of the non-dominated solutions obtained by the MOWFA cover the final Pareto set (see the corresponding instances in the Ψ_3 metric column of MOWFA in Tables 6.2 and 6.3). From the average values of metrics Ψ_1, Ψ_2, Ψ_3, and Ψ_4 in Tables 6.2 and 6.3, it can also be seen that the MOWFA obtained more

diversified and competitive Pareto sets than IHMOPGA in both cases of no buffer and finite buffers.

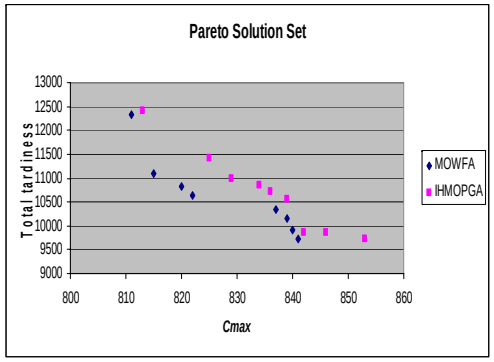
Table 6.2 Comparison of MOWFA and IHMOPGA for the Wittrock Benchmarks with No Buffer

Instance	$N \times S$	MOWFA				IHMOPGA			
		Ψ_1	Ψ_2	Ψ_3	Ψ_4	Ψ_1	Ψ_2	Ψ_3	Ψ_4
1	51×3	8.00	0.00	100.00	0.00	9.00	100.00	0.00	13.67
2	38×3	6.00	0.00	100.00	0.00	5.00	100.00	0.00	24.79
3	38×3	7.00	0.00	100.00	0.00	6.00	100.00	0.00	25.85
4	36×3	6.00	0.00	100.00	0.00	5.00	80.00	16.67	18.26
5	40×3	9.00	0.00	100.00	0.00	9.00	100.00	0.00	10.17
6	30×3	8.00	12.50	87.50	2.24	7.00	85.71	12.50	10.87
Average		7.33	2.08	97.92	0.37	6.83	94.29	4.86	17.27

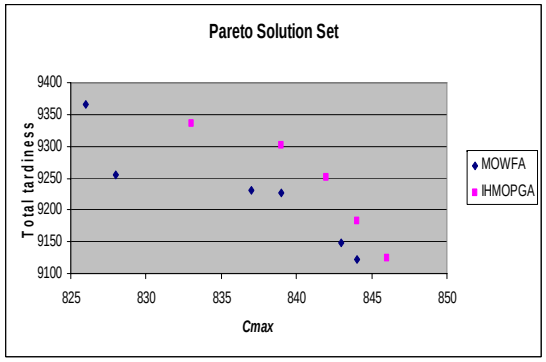
Table 6.3 Comparison of MOWFA and IHMOPGA for the Wittrock Benchmarks with Finite Buffers

Instance	$N \times S$	MOWFA				IHMOPGA			
		Ψ_1	Ψ_2	Ψ_3	Ψ_4	Ψ_1	Ψ_2	Ψ_3	Ψ_4
1	51×3	7.00	0.00	100.00	0.00	8.00	87.50	14.29	16.46
2	38×3	14.00	28.57	76.92	0.86	11.00	72.73	23.08	6.65
3	38×3	8.00	12.50	70.00	1.78	8.00	62.50	30.00	5.23
4	36×3	9.00	0.00	90.00	1.43	9.00	88.89	10.00	8.36
5	40×3	4.00	0.00	100.00	0.00	4.00	75.00	25.00	12.5
6	30×3	7.00	0.00	100.00	0.00	8.00	87.50	14.29	9.38
Average		8.17	6.85	89.49	0.68	8.00	79.02	19.44	9.76

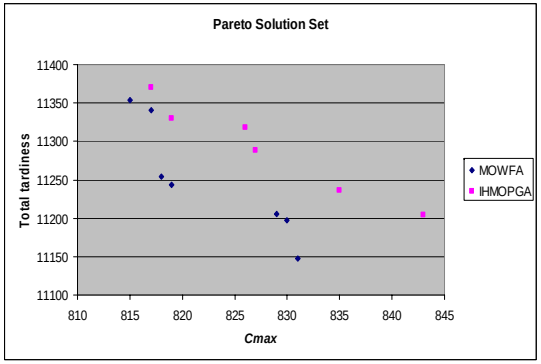
We further illustrate the comparison results between MOWFA and IHMOPGA for Wittrock instances with no buffer and finite buffers in Figures 6.8 and 6.9, respectively. It can be seen that the solutions obtained by the MOWFA cover more area of the Pareto set than those obtained by the IHMOPGA. Also, the solutions of the MOWFA are closer to the Pareto set than that of IHMOPGA.



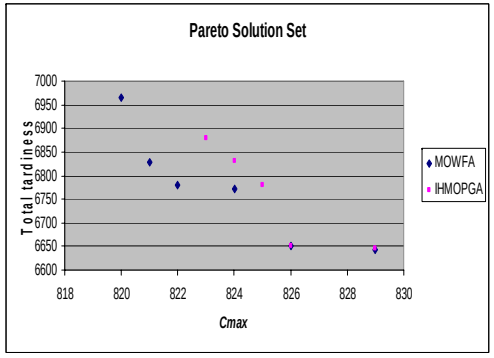
a) Instance 1



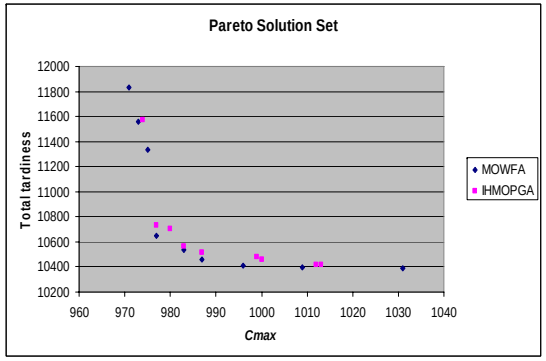
b) Instance 2



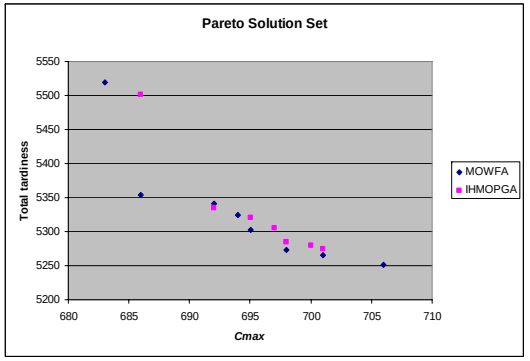
c) Instance 3



d) Instance 4

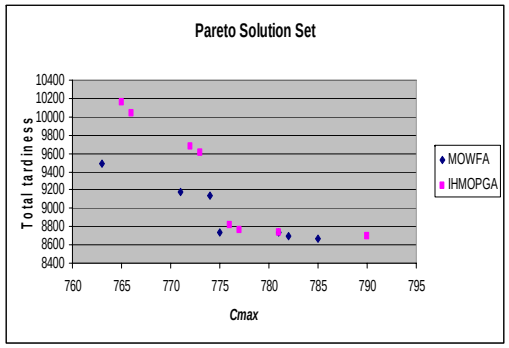


e) Instance 5

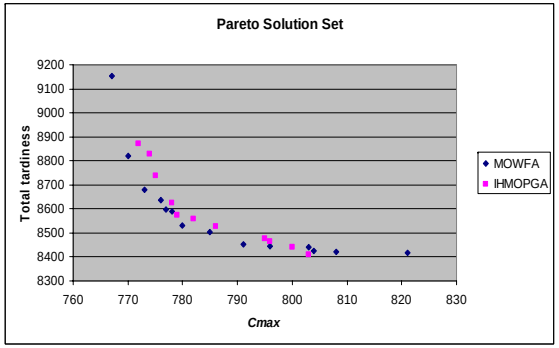


f) Instance 6

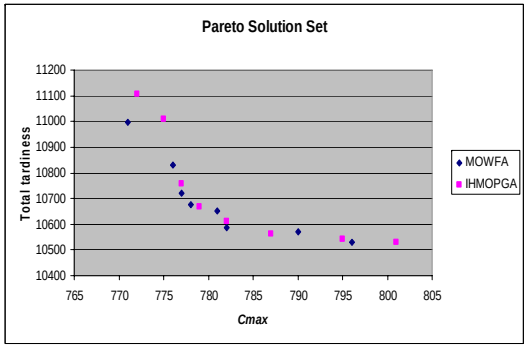
Figure 6.8 Plot of Pareto Fronts Obtained by MOWFA and IHMOPGA on Wittrock Instances with No Buffer



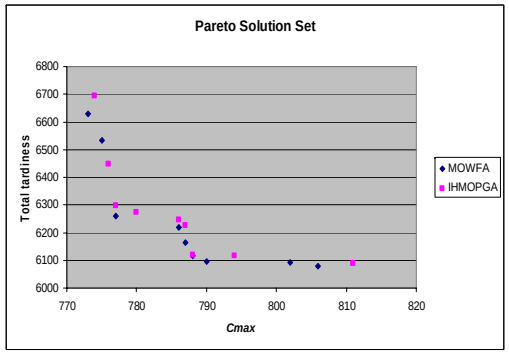
a) Instance 1



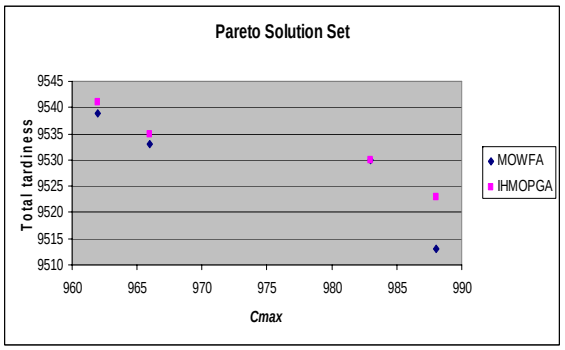
b) Instance 2



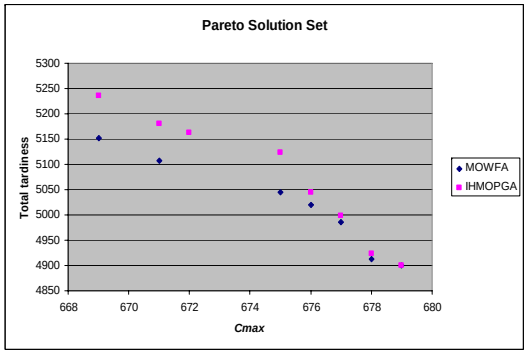
c) Instance 3



d) Instance 4



e) Instance 5



f) Instance 6

Figure 6.9 Plot of Pareto Fronts Obtained by MOWFA and IHMOPGA on Wittrock Instances with Finite Buffers

The comparison results for the randomly generated instances are shown in Tables 6.4 and 6.5. Here, Table 6.4 shows the results of MOWFA and IHMOPGA for the generated instances with no available buffer capacity, while Table 6.5 shows the results for the generated instances with finite buffers.

From Tables 6.4 and 6.5, we can see that the MOWFA still outperforms IHMOPGA in all instances. The average values of the metrics Ψ_1, Ψ_2, Ψ_3 and Ψ_4 in Tables 6.4 and 6.5 show that the MOWFA can obtain more diversified and competitive Pareto sets than the IHMOPGA. For the case of no buffer in Table 6.4, although the number of non-dominated solutions of MOWFA is less than that of IHMOPGA, they cover 96.8% of the area of the Pareto set, while the solutions of IHMOPGA only cover 9.6% of the area of the Pareto set. Moreover, 88.72% of the solutions obtained by the IHMOPGA are dominated by that of MOWFA, while the metric value of MOWFA is only 2.55%. In addition, the distance of the solutions obtained by MOWFA to the Pareto set is much closer than that of IHMOPGA, i.e., 0.48 and 11.93 for MOWFA and IHMOPGA, respectively.

For the case of finite buffer in Table 6.5, although the quality of non-dominated solutions obtained by the IHMOPGA has improved, they are still outperformed by the non-dominated solutions obtained by the MOWFA. In particular, the dominated solutions of the IHMOPGA have reduced to 75.72%, and they covered 17.66% of the area of the Pareto set. They are thus worse than the non-dominated solutions obtained by the MOWFA.

Through these comparison results, it can be concluded that the MOWFA outperforms IHMOPGA for the instances tested, and the MOWFA is thus an efficient algorithm for solving the MOFFSP with intermediate buffers.

Table 6.4 Comparison of MOWFA and IHMOPGA for the Randomly Generated Instances with No Buffer

Instance	$N\times S$	MOWFA				IHMOPGA			
		Ψ_1	Ψ_2	Ψ_3	Ψ_4	Ψ_1	Ψ_2	Ψ_3	Ψ_4
1	20×2	8.80	0.00	100.00	0.00	10.80	86.91	11.16	4.56
2	40×2	13.80	0.00	100.00	0.00	13.20	88.02	11.65	5.66
3	60×2	3.40	0.00	100.00	0.00	4.80	96.00	5.00	11.22
4	80×2	4.80	0.00	100.00	0.00	4.80	96.00	5.00	13.23
5	100×2	5.20	0.00	100.00	0.00	4.40	96.00	4.00	27.88
6	20×4	11.20	0.00	100.00	0.00	12.40	90.23	10.76	8.06
7	40×4	11.00	25.45	82.00	1.66	11.80	74.55	18.00	6.41
8	60×4	12.20	0.00	100.00	0.00	11.20	82.12	16.41	10.31
9	80×4	7.20	0.00	86.00	3.13	5.20	77.33	14.00	22.71
10	100×4	5.20	0.00	100.00	0.00	6.80	100.00	0.00	9.26
Average		8.28	2.55	96.80	0.48	8.54	88.72	9.60	11.93

Table 6.5 Comparison of MOWFA and IHMOPGA for the Randomly Generated Instances with Finite Buffers

Instance	$N\times S$	MOWFA				IHMOPGA			
		Ψ_1	Ψ_2	Ψ_3	Ψ_4	Ψ_1	Ψ_2	Ψ_3	Ψ_4
1	20×2	7.80	0.00	100.00	0.00	7.20	60.15	22.86	15.45
2	40×2	7.80	0.00	100.00	0.00	6.20	100.00	0.00	7.66
3	60×2	3.20	0.00	100.00	0.00	3.00	33.33	63.34	0.55
4	80×2	4.80	0.00	100.00	0.00	2.40	93.33	5.00	49.35
5	100×2	2.20	0.00	100.00	0.00	2.00	100.00	0.00	74.79
6	20×4	6.20	0.00	86.07	0.69	7.20	68.72	13.93	12.08
7	40×4	7.20	13.93	75.56	1.64	5.00	60.00	24.44	17.32
8	60×4	7.80	0.00	78.00	1.30	6.00	66.67	22.00	10.98
9	80×4	4.00	0.00	100.00	0.00	4.00	75.00	25.00	20.20
10	100×4	3.00	0.00	100.00	0.00	4.00	100.00	0.00	51.95
Average		5.40	1.39	93.96	0.36	4.70	75.72	17.66	26.03

6.5 Conclusions

In this chapter, a MOWFA is proposed for solving the MOFFSP with intermediate buffers. To solve the problem by the proposed algorithm, we divide the feasible solution set into multiple distinct regions. Then landscape analysis is performed for each region to determine the weights for the objectives, which guide the *DOW*s to search for local optimal solutions in the corresponding region. Two erosion schemes are proposed in the MOWFA. The first scheme focuses on exploiting the current region, while the second scheme helps *DOW*s exploit other potential regions. We also incorporate an evaporation process in the MOWFA to enhance the exploitation capability in potential neighboring regions. The Pareto solution set obtained is reinforced by an improvement process.

We used the Wittrock benchmark instances and the randomly generated instances to evaluate the performance of the MOWFA. The proposed algorithm is compared with other algorithms that can be applied to solve the MOFFSP problem. The comparison results show that the MOWFA outperforms the other algorithms for the test instances. These results have been reported in Tran and Ng (2011[a]).

The MOWFA can be applied to solve the MOFFSP with a large number of objectives. However, if we use the ellipsoid approximation method to determine the relative weights of the objectives, the MOWFA can only successfully solve the MOFFSP with at most three objectives. For the MOFFSP in which the number of objectives is greater than three, we need to use a linear regression method to determine the weights.

CHAPTER 7

WFA FOR OTHER COMBINATORIAL OPTIMIZATION PROBLEMS

In this chapter, WFA is used to solve two other well-known combinatorial optimization problems which are the quadratic assignment problem (QAP) and vehicle routing problem (VRP). The QAP and VRP are NP-hard optimization problems often encountered in facility layout design (Sahni, 1976) and the field of logistics and supply chain management (Toth and Vigo, 2002), respectively.

For the QAP, a systematic precipitation generating scheme is included in the WFA for spreading raindrop positions on the ground to increase the solution exploration capability of the algorithm. Efficient local search methods are also used to enhance the solution exploitation capability of this algorithm. The performance of the proposed algorithm is tested with the benchmark instances taken from the QAPLIB (Burkard et al., 1997). Computational results and comparisons show that the WFA is able to obtain good quality or optimal solutions to the QAP instances in reasonable computation time.

For the VRP, we have developed a two-level WFA (2LWFA). The first level of the proposed algorithm is to solve the mathematical programming model of the VRP with the

relaxation of the integrality constraints. At the second level, a modified WFA is then applied to search for optimal solutions from the initial solutions obtained from the first level. Here, we illustrate the performance of the 2LWFA with the capacitated vehicle routing problem (CVRP). Some preliminary computational results show the efficiency of this algorithm for the VRP.

Chapter 7 is organized as follows. In Section 7.1, we present the WFA proposed for solving the QAP. In particular, we introduce the QAP and its important applications. A literature review of solution methods for the problem is also described. Then, the implementation of WFA for solving the QAP is presented. Computational experiments and comparisons based on the QAP benchmark instances are shown in this section. In Section 7.2, we present the 2LWFA proposed for solving the CVRP. The CVRP is first introduced and a description of the 2LWFA is then provided. The results of preliminary experiments carried out on the CVRP benchmark instances from the literature are also shown. Finally, some conclusions of this chapter are provided in Section 7.3.

7.1 **Quadratic Assignment Problem**

7.1.1 Introduction

The QAP is one of the well-known NP-hard optimization problems (Sahni, 1976) due to its important applications in practice, such as parallel and distributed computing (Bokhari, 1987), statistical data analysis (Hubert, 1987), testing of electronic devices (Eschermann and Wunderlich, 1990), plant layout design (Rossin et al., 1999), data visualization (Abbiw-Jackson et al, 2006), printed circuit board assembly process (Duman and Or,

2007), and website structure improvement (Qahri Saremi et al., 2008). The problem is often described as follows: Given a set of facilities and a set of locations with the same size n, assign the facilities to the locations such that the total cost of assignment is minimized. The total cost w is calculated using the distance between locations and the flow between facilities. The QAP can then be expressed as the problem of finding a permutation π of n facilities as follows:

$$\min_{\pi \in \Pi_n} w(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}\, d_{\pi[i]\pi[j]}, \tag{7.1}$$

where Π_n denotes the set of possible permutations of $N = \{1, 2, \ldots, n\}$, while $\pi[i]$ and $\pi[j]$ denote the location of facilities i and j in the permutation π, respectively. Furthermore, f_{ij} is the flow between facilities i and j, and $d_{\pi[i]\pi[j]}$ is the distance between locations $\pi[i]$ and $\pi[j]$.

While there are some well solved special cases of the QAP (see for example, Demidenko et al. (2006)), the QAP is generally difficult to solve (Taillard, 1995), and so researchers have explored various possible solution methods. These solution methods can be broadly classified into exact methods and heuristic methods. Exact methods aim to find the optimal solution to the QAP but they often run into computational difficulties with large QAP instances (Burkard et al., 1996; Cela, 1998; Hahn et al., 2001; Blanchard et al., 2003; and Gasimov and Ustun, 2007). Heuristic methods on the other hand can be further divided into constructive heuristics and improvement heuristics. Constructive heuristics often construct a feasible solution for the QAP by assigning each facility to a location according to some principles. Such constructive methods for solving the QAP can be seen

in Buffa et al. (1964), Arkin et al. (2001), and Gutin and Yeo (2002). Unlike constructive heuristics, improvement heuristics attempt to improve an existing solution through some iterative procedures. Some of the well-known algorithms belonging to this category are the meta-heuristic algorithms, such as genetic algorithms (Ahuja et al., 2000; and Lim et al., 2002), ant colony optimization algorithms (Maniezzo and Colorni , 1999; Ramkumar et al., 2009; and Wong and See, 2010), greedy randomized adaptive search procedure (Li et al., 1994), memetic algorithm (Merz and Freisleben, 2000), iterated fast local search algorithm (Ramkumar et al., 2008), simulated annealing algorithm (Singh and Sharma, 2008), and DNA algorithm (Yang et al., 2008).

In this chapter, we develop the WFA for solving the QAP. It uses distributed drops of water (*DOW*s) to represent the permutations in the QAP. Benchmark problem instances drawn from the QAPLIB (Burkard et al., 1997) are used to evaluate the performance of the proposed algorithm. The computational results and comparisons show that the WFA is able to obtain good quality or optimal solutions to these instances and is a promising nature-inspired algorithm for solving QAP problem instances.

7.1.2 WFA for the QAP

In this section, we describe the procedure of solving the QAP by the WFA in detail. The main phases in the WFA include the exploration phase, the exploitation phase, and the improvement phase, as well as a systematic precipitation generating scheme. We first begin with a description of the basic components for the WFA.

7.1.2.1 Encoding Scheme and Memory Lists

For the QAP, we consider the permutation π of n facilities in the problem as the longitude and latitude in the position of *DOW* on the ground, while the total cost of flow between the facilities is encoded as the altitude. Given a permutation of n facilities $\pi = (\sigma_1, ..., \sigma_n)$, we thus define:

$$\text{longitude}(\pi) = (\sigma_1, ..., \sigma_{\lfloor \frac{n}{2} \rfloor}), \tag{7.2}$$

$$\text{latitude}(\pi) = (\sigma_{\lfloor \frac{n}{2} \rfloor + 1}, ..., \sigma_n), \tag{7.3}$$

where $\lfloor q \rfloor$ is the largest integer less than or equal to q. Figure 7.1 shows an illustrative example of a *DOW* and its positional vector components for the QAP with $n = 8$ facilities.

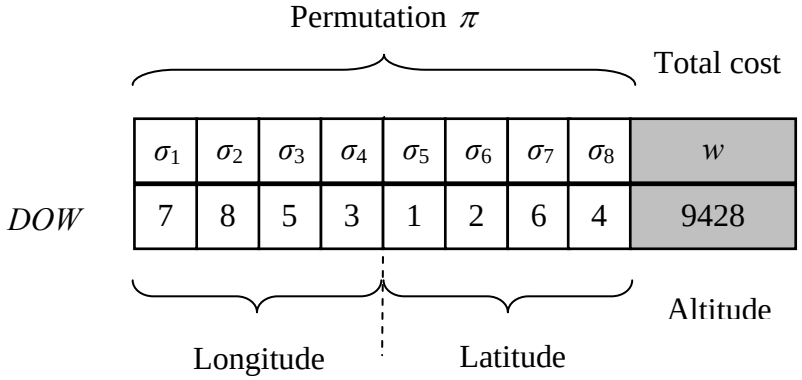


Figure 7.1 An Example of Solution Representation in the WFA for the QAP

With this encoding scheme, the neighborhood structure used in the WFA for the QAP is mainly based on exchanging elements in a permutation. An example is the 2-opt neighborhood structure used in the traveling salesman problem as well as in the QAP

(Merz and Freisleben, 2000). Thus if π' is the permutation obtained by exchanging positions of two facilities i and j in the permutation π, we can determine π' by:

$$\begin{aligned} &\pi'[k] = \pi[k] \quad \text{for} \quad k \in N \setminus \{i, j\}, \\ &\pi'[i] = \pi[j], \\ &\pi'[j] = \pi[i]. \end{aligned} \tag{7.4}$$

In addition, we also consider the neighborhood structure of an extended 2-opt algorithm called the 2-opt mirror. In this 2-opt mirror algorithm, other than the usual 2-opt neighboring permutations, we also consider the reflected permutation and its corresponding 2-opt neighboring permutations. The reflected permutation π_r of a permutation π can be determined as follows:

$$\pi_r[i] = \pi[n - i + 1] \quad \text{for } i=1,..,n. \tag{7.5}$$

The neighborhood structures are used for both exploration and exploitation phases.

To support the search for global optimal permutations, three sets of memory lists, namely the best permutations list (P_0-list), the un-eroded permutations list (UE-list), and the eroded permutations list (E-list), have been adopted to solve the QAP. Here the functions of the lists and the procedure of updating these lists are similar as that described in Chapter 4.

Let π_c^{Best} be the best optimal permutation found so far by the WFA at cloud c. Assume that we have found a local optimal permutation π_{c+1}^* at cloud $c + 1$. Updating the P_0-list at the cloud can then be described as follows:

$$\mathrm{P}_0\text{-list} = \begin{cases} \pi_{c+1}^{*} & \text{if} \quad w(\pi_{c+1}^{*}) < w(\pi_c^{Best}), \\ \pi_c^{Best} & \text{otherwise.} \end{cases} \tag{7.6}$$

Updating of the UE-list includes two phases, the phase of removing the local optimal permutations eroded and the phase of adding the ones just found, which are done in succession. Updating of the E-list is only to add the local optimal permutations eroded. They are shown as follows:

$$\text{UE-list}_{c+1} = \left(\text{UE-list}_c \cup \Pi_c^2\right) \setminus \Pi_c^1, \tag{7.7}$$

$$\text{E-list}_{c+1} = \text{E-list}_c \cup \Pi_c^1. \tag{7.8}$$

where Π_c^1 and Π_c^2 denote the set of local optimal permutations eroded and the set of local optimal permutations just found from initial permutations generated at each iteration, respectively.

7.1.2.2 Exploration Phase

Two schemes for generating the initial population of *DOW* positions in the exploration phase are proposed below.

The first scheme is the random permutation generator scheme. In this scheme, a population Ω of permutations of *DOW*s is generated randomly for each cloud and the number of such permutations is the maximum population size allowed (*MaxPop*). Since the WFA mimics the property of water flow always moving from higher positions to lower positions due to Earth's gravity, a steepest descent hill sliding algorithm is then

applied to search for local optimal permutations from these initial permutations. In particular, from an initial solution, the hill sliding algorithm searches for the best improved solution within the initial solution's neighbors in terms of objective value. Then, this process continues to be performed iteratively for the improved solution obtained until no other improved solution is found.

Due to the random nature of this scheme, the efficiency of WFA for solving the QAP may fluctuate in instances with large size. To resolve this drawback of the first scheme as well as to improve the solution exploration capability of WFA for QAP instances with large size, a second scheme is proposed. This scheme is a systematic *DOW* generator scheme that aims to distribute *DOW*s evenly into divided regions of the solution space. We first divide the solution space into *n* regions, where *n* is the instance size. Then at each cloud, the WFA generates *n DOW*s and each *DOW* is assigned to only one distinct region. This means that a cloud would consist of *n* different permutations, which is achieved by fixing the first position of a facility in a permutation from 1 to *n* when generating the permutations of *DOW*. The rest of the positions in a permutation are assigned randomly. This can be represented with the following notation:

$$\Omega_c = \left\{ \left[\pi_c^1, ..., \pi_c^i, ..., \pi_c^n \right]^{\mathrm{T}} \quad \middle| \quad \pi_c^i[1] = i, (i = 1, ..., n) \right\}, (c = 1, ..., MaxCloud). \quad (7.9)$$

A steepest descent hill sliding algorithm is also used in the second scheme to search for local optimal permutations from these initial permutations.

At each cloud, the number of *DOW*s at the initial permutations is updated as follows:

$$Q_{c+1}^{\pi_{c+1}^i} = \begin{cases} Q_c^{\pi} + 1 & \text{if} \quad \pi_{c+1}^i = \pi \in \Pi_c^3, \\ 1 & \text{otherwise,} \end{cases} \quad \text{for} \quad i = 1, ..., MaxPop, \tag{7.10}$$

where Q_c^{π} is the number of *DOW*s in permutation π at cloud c, and π_{c+1}^i denotes the permutation of the ith *DOW* generated at cloud $c + 1$, while the set of initial permutations generated or local optimal permutations found through clouds 1 to c is denoted by Π_c^3.

After the steepest descent hill sliding algorithm has been applied to find the local optimal permutation π_{c+1}^* from the initial permutation π_{c+1}^i, we also update the number of *DOW*s at the optimal permutation according to the following equation.

$$Q_{c+1}^{\pi_{c+1}^*} = \begin{cases} Q_c^{\pi_{c+1}^*} + Q_{c+1}^{\pi_{c+1}^i} & \text{if} \quad \pi_{c+1}^* \in \text{UE-list}, \\ Q_{c+1}^{\pi_{c+1}^i} & \text{otherwise.} \end{cases} \tag{7.11}$$

In general, the solution exploration phase of the WFA for the QAP results in a set of local optimal permutations. They are updated in the UE-list to be considered for the erosion process in the next exploitation phase.

7.1.2.3 Exploitation Phase

The exploitation phase involves applying the erosion process to overcome the local optimal permutations found in the exploration phase. Before describing the erosion process, the erosion condition and capability are first described below:

7.1.2.3.1 Erosion Condition and Capability

The erosion process is triggered by the amount of precipitation and so if the number of *DOW*s at a local optimal permutation increases to the threshold *MinEro*, the erosion process is performed at this local optimal permutation.

Next, we consider the capability L of the erosion process. For the QAP, the erosion capability is based on a factor, which is the number of *DOW*s at the eroding local optimal permutation. In particular, the relationship between the erosion capability and this factor is a nonlinear function. However, to simplify the computations in the WFA, we have set the erosion capability to a constant value *MaxUIE*, so that the relationship can be described as follows:

$$L\left(Q_c^{\pi_c^*}\right) = \begin{cases} MaxUIE & \text{if} \quad Q_c^{\pi_c^*} \geq MinEro, \\ 0 & \text{otherwise.} \end{cases} \tag{7.12}$$

If the erosion process cannot find any improved permutation after *MaxUIE* search steps, the erosion process stops and other permutations in the UE-list are considered for performing the next erosion process.

7.1.2.3.2 Erosion Process

The erosion process is the main operator in the exploitation phase of WFA for solving the QAP. Its task is to help the *DOW*s overcome local optimal permutations and obtain better local optimal or global optimal permutations. In the QAP, the erosion process depends on a topological parameter Δd_h representing the geographical shape of the surface. It is

defined as the difference of total cost between the local optimal permutation and its *h*th neighboring permutation:

$$\Delta d_h = w\left(\pi_h^*\right) - w\left(\pi^*\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}\left(d_{\pi_h^*[i]\pi_h^*[j]} - d_{\pi^*[i]\pi^*[j]}\right) \quad \text{for} \quad h = 1,\ldots,\frac{n(n-1)}{2}, \tag{7.13}$$

where π^* and π_h^* denote the local optimal permutations and its *h*th neighboring permutation respectively. In the case of the 2-opt mirror neighborhood, the number of directions is $n(n-1)+1$.

The aim of computing Δd_h is to help the erosion process choose the most suitable direction to perform erosion. The erosion process will choose the smallest Δd_h to be the first erosion direction. Searching for this direction will stop if the erosion process for the erosion direction cannot find a better permutation after *MaxUIE* steps. Then, the erosion process will be restarted with another erosion direction with the next smallest Δd_h. If the erosion process cannot find a better permutation for all the directions, we move the eroded local optimal permutation into the E-list so that it will not be considered for erosion process in the next clouds. On the other hand, if the erosion process is able to find a better permutation than the eroding local optimal permutation, that erosion direction is chosen to erode the local optimal permutation permanently. Then the new local optimal permutation is updated in the UE-list to continue with performing the erosion process.

7.1.2.4 Improvement Phase

The exploration and exploitation phases of the WFA for the QAP terminate when the maximum number of clouds (*MaxCloud*) has been generated. To improve the solutions obtained by the WFA, we can apply the 3-opt algorithm at a final improvement phase. This 3-opt algorithm would still use the steepest descent hill sliding method mentioned in the previous section. A 3-opt list is also used to save the solutions obtained by this improvement algorithm.

In summary, we have the following variants of WFA:

(1). Random 2-opt WFA: WFA with the random permutation generator scheme and the 2-opt neighborhood structure.

(2). Systematic generator 2-opt WFA: WFA with the systematic *DOW* generator scheme and the 2-opt neighborhood structure.

(3). Random 2-opt mirror WFA: WFA with the random permutation generator scheme and the 2-opt mirror neighborhood structure.

A flow chart of the WFA for solving the QAP is shown in Figure 7.2.

7.1.3 Computational Experiments and Comparisons

To test the performance of the proposed WFA, we have used the random 2-opt WFA for solving all the benchmark instances from the QAPLIB (Burkard et al., 1997). The systematic generator 2-opt WFA and the random 2-opt mirror WFA are used to solve the benchmark instances when the random 2-opt WFA has not obtained the best known value.

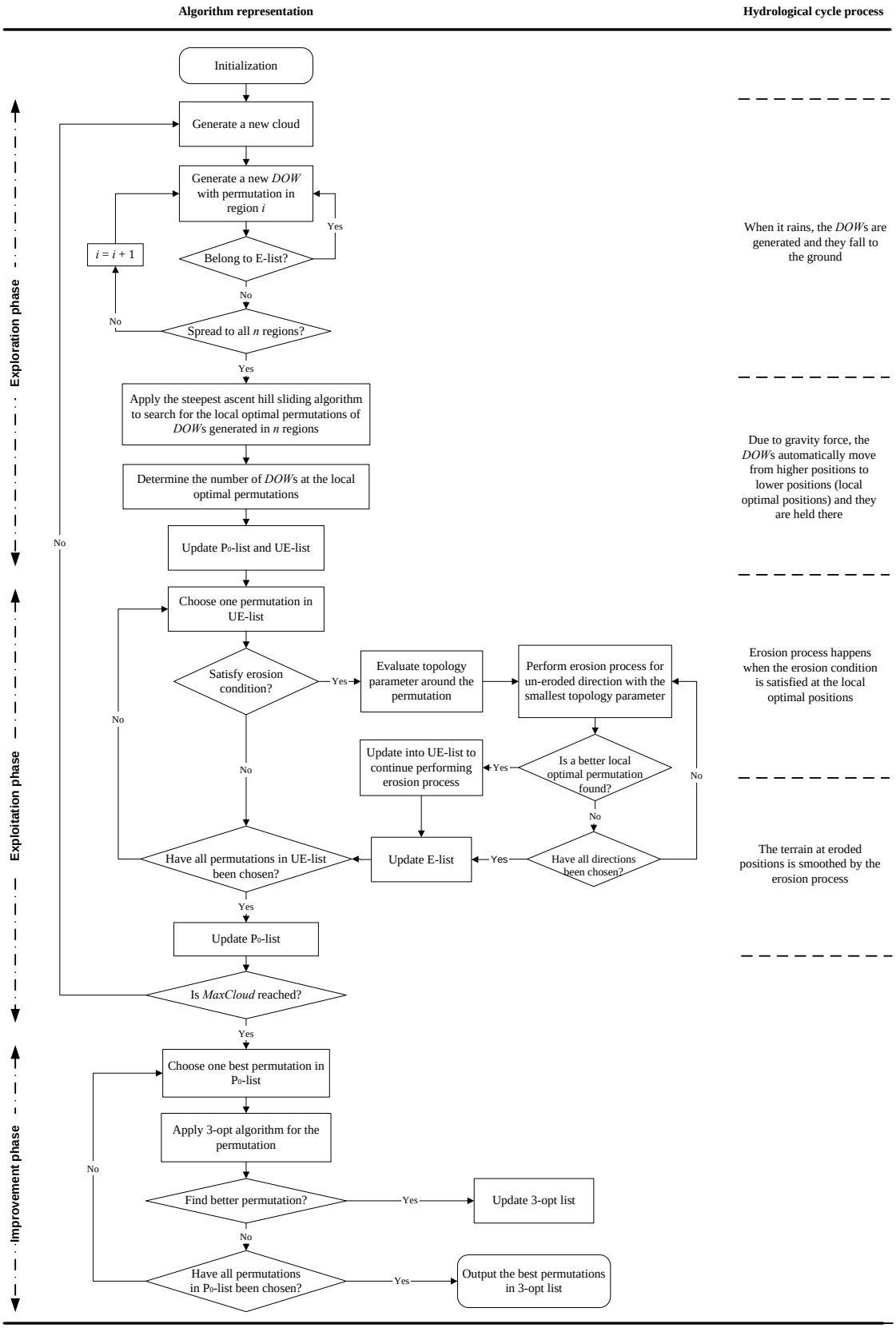


Figure 7.2 Flow Chart of the WFA for the QAP

7.1.3.1 Benchmark Problem Sets

The 134 instances drawn from the QAPLIB (Burkard et al., 1997) are well-known benchmark problem sets in QAP with size ranging from 12 to 256. The best known upper bounds of these problem sets obtained from the literature were used to compare with the best results obtained by the WFA. In addition, we also compared these results with those obtained by greedy randomized adaptive search procedure (GRASP) of Li et al. (1994), by ant system (ANT) of Maniezzo and Colorni (1999), by greedy genetic algorithm (GGA) of Ahuja et al. (2000), by hybrid genetic algorithm with partial local search (PGA) of Lim et al. (2002), by iterated fast local search algorithm (IFLS) of Ramkumar et al. (2008), by two-level modified simulated annealing based approach (MSA) of Singh and Sharma (2008), and by population-based hybrid ant system (PHAS) of Ramkumar et al. (2009), all of which are some of the most efficient meta-heuristic algorithms for solving the QAP instances. Almost all the algorithms used the design-of-experiment method or trial runs to select their best parameter values. However, the number of runs of each algorithm can be different, such as 5 runs used for the GRASP, ANT, and PHAS, 7 runs used for GGA, and 10 runs for PGA. Also, some algorithms did not provide information on the number of runs used, such as IFLS and MSA. Thus, to have a fair comparison, we used the smallest number of runs that several algorithms have used, i.e., 5 runs.

7.1.3.2 Platform and Parameters

All the computational experiments were performed on an Intel Centrino Duo 1.60 GHz CPU with 1.5 GB of RAM. The WFA has been coded using Microsoft Visual Basic 6.0. Here, the computational complexity of the WFA for the QAP is determined based on the

neighborhood structures used and the erosion process of this algorithm. In particular, the WFA used 2-opt neighborhood structure, and the worst possibility of the erosion process is to find for all n directions. Hence, the computational complexity of the WFA is estimated to be $O(n^3)$. Although the WFA for the QAP used 3-opt neighborhood structure at the improvement phase, since the 3-opt algorithm is only used once, the computational complexity of the WFA for the QAP is still $O(n^3)$.

The choice of parameters for WFA was determined by the design-of-experiment method. In particular, we have carried out several simulations that test the WFA on all types of QAP with various values for the controlled parameters, i.e., the exploration parameters *MaxCloud* and *MaxPop*, and the exploitation parameters *MaxUIE* and *MinEro*. The aim is to determine the best parameters of WFA for QAP that would achieve a balance between solution exploration and exploitation capabilities in finding the best solutions within reasonable computation time. Thus, smaller values may be used by the parameters for larger problem instances. The values that were used are as follows: *MaxCloud* = 2, 5, 10, 15, 20; *MaxPop* = 5, 10, 15, 20; *MaxUIE* = 5, 10, 15, 20; and *MinEro* = 2, 3. From preliminary simulation results, the best parameter sets are shown in Table 7.1. These parameter sets are then used for the random 2-opt WFA and the random 2-opt mirror WFA. For the systematic generator 2-opt WFA, we have used the parameter sets (*MaxCloud*, *MaxPop*, *MaxUIE*, *MinEro*) = (50, *n*, 20, 2), (20, *n*, 10, 2), and (5, *n*, 5, 2) for instances with size at most 50, more than 50 but at most 100, and more than 100 respectively, in order to allow a reasonable amount of erosion process to occur. With these parameter sets, 5 independent replicates were used for each instance and the best results were used to compare the WFA with other meta-heuristic algorithms.

Table 7.1 Parameter Sets of WFA for the QAP Benchmark Instances

Benchmarks	*n*	Parameter values			
		MaxCloud	*MaxPop*	*MaxUIE*	*MinEro*
Burkard	26	5	10	10	2
Christofides	12 – 20	10	10	10	2
	22	15	10	10	2
	25	20	10	10	2
Elshafei	19	10	10	10	2
Eschermann	16, 64	2	5	5	2
	32 (a, b)	5	10	10	2
	32 (c, e, g)	2	5	5	2
	32 (d, h)	2	10	10	2
	128	5	10	5	3
Hadley	12 – 20	10	10	10	2
Krarup	30, 32	20	10	10	2
Li & Pardalos	20, 30	10	10	10	2
	40, 50, 60	10	20	10	2
	70	10	20	15	2
	80, 90	20	20	10	2
Nugent	12 – 28	10	10	5	2
	30	20	10	10	2
Roucairol	12, 15	5	10	10	2
	20	10	20	10	2
Scriabin	12, 15, 20	5	15	10	2
Skorin-Kapov	42 – 64	15	10	10	2
	72, 81, 90	10	20	15	2
	100	10	10	10	2
Steinberg	36	20	10	10	2
Taillard					
(*Taixxxa*)	12	5	10	10	2
	15, 17	5	20	10	2
	20 – 35	20	20	10	2
	40, 50	20	20	15	2
	60, 80, 100	10	20	10	2
(Taixxxb)	12 – 20	5	5	10	2
	25	10	10	10	2
	30, 35, 40	20	10	10	2
	50, 60, 80	10	10	15	2
	100	5	20	10	2
	150	5	10	5	2
(Taixxxc)	64	10	20	10	2
	256	2	5	5	2
Thonemann	30	10	20	10	2
	40	20	10	10	2
	150	5	10	10	2
Wilhelm	50	15	20	10	2
	100	10	10	10	2

7.1.3.3 Performance Measures

For comparison of objective values, we have used the following relative percentage difference in objective value:

$$\Delta_{Best} = \left(\frac{Heuristic_{sol} - Opt_{sol}}{Opt_{sol}} \right) \times 100, \tag{7.14}$$

where $Heuristic_{sol}$ and Opt_{sol} denote the best objective function value obtained by the WFA and the best known value in the literature, respectively.

The best known values from the literature are used as the reference values for the evaluation and comparison of the WFA and other algorithms for solving the QAP. These values may be obtained from the optimal solutions of the benchmark instances used, or the best solutions found by some algorithm so far. Thus, the comparison results may be negative values if the WFA finds a better solution.

For evaluation and comparison of the overall performance of algorithms, we have used criteria such as the average relative percentage difference for all instances solved and the number of the best known solutions obtained in all instances solved. Thus, the algorithm with a smaller average relative percentage difference and a larger number of the best known solutions obtained would be considered to be a more effective optimization method.

7.1.3.4 Computational Results

The computational results and comparisons with other meta-heuristic algorithms, such as GRASP, ANT, GGA, PGA, IFLS, MSA, and PHAS, are shown in Tables 7.2a to 7.2e, while the improvement results obtained by the variants of WFA for the QAP benchmark instances that have not been solved optimally by the 2-opt WFA are displayed in Table 7.3. In Tables 7.2a to 7.2e, the column with the best results of WFA shows the best solutions obtained by the random 2-opt WFA and the variants of WFA. Since the details of the computation time of applying PHAS to the QAP instances were not shown in Ramkumar et al. (2009), we did not include this information in Tables 7.2a to 7.2e. In addition, the symbol "—" in the entries of these tables is used to indicate that the compared algorithms have not solved the respective benchmark instances. For Table 7.3, the entries displayed in italic font highlight the best results obtained by the respective variant of the WFA.

From Table 7.3, we can see that the systematic generator 2-opt WFA, the random 2-opt mirror WFA and the WFA-3-opt have obtained better results than the random 2-opt WFA for some of the instances when the random 2-opt WFA was unable to obtain the best known solution. In particular, these variants improved the solution quality for 25 instances.

From the best results obtained by the WFA in Tables 7.2a to 7.2e, it can be seen that out of the 134 instances from the QAPLIB (Burkard et al., 1997), the best known solutions for 99 instances have been obtained by the WFA within reasonable computation time. The WFA is also able to obtain solutions with a relative percentage difference of less than 2% for all the remaining instances. The average relative percentage difference of WFA for all the 134 instances is found to be 0.20%.

Table 7.2a Comparison Results of the WFA with Other Algorithms for Burkard's and Christofides' Instances

Instances	Best known value	Random 2-opt WFA		Best results of WFA		GRASP		ANT		GGA		PGA		IFLS		MSA		PHAS
		Best solution	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}
Bur26a	5426670	5426670	140.0	**0**	140.0	0	11.38	0	21.07	0	235	0	125	0	61.27	—	—	0
Bur26b	3817852	3817852	108.0	**0**	108.0	0	59.45	0	35.03	0	225	0	9.5	0	60.27	—	—	0
Bur26c	5426795	5426795	80.0	**0**	80.0	0	5.16	0	19.09	0	227	0	7.42	0	57.78	—	—	0
Bur26d	3821225	3821225	126.0	**0**	126.0	0	15.12	0	19.4	0	213	0	8.42	0	61.27	—	—	0
Bur26e	5386879	5386879	145.0	**0**	145	0	17.63	0	20.53	0	218	0	10.03	0	57.83	—	—	0
Bur26f	3782044	3782044	197.0	**0**	197	0	5.05	0	11.23	0	204	0	6.68	0	59.19	—	—	0
Bur26g	10117172	10117172	129.0	**0**	129	0	22.58	0	18.67	0	194	0	9.99	0	57.72	—	—	0
Bur26h	7098658	7098658	97.0	**0**	97	0	37.58	0	5.67	0	204	0	6.82	0	57.47	—	—	0
Chr12a	9552	9552	2.1	**0**	2.1	—	—	—	—	0	19.6	0	0.54	0	1.09	0	40	0
Chr12b	9742	9742	1.8	**0**	1.8	—	—	—	—	0	18.4	0	0.42	0	1.11	0	41	0
Chr12c	11156	11156	2.2	**0**	2.2	—	—	—	—	0	20.2	0	1.29	0	1.02	0.26	38	0.27
Chr15a	9896	9896	11.4	**0**	11.4	—	—	—	—	0.4	40.6	0	1.5	0	2.97	0	69	0
Chr15b	7990	7990	15.4	**0**	15.4	—	—	—	—	0	41.8	0	1.31	0	3.08	2.7	72	0
Chr15c	9504	9504	14.3	**0**	14.3	—	—	—	—	0	44	0	1.30	0	2.64	11.5	69	6.36
Chr18a	11098	11098	40.0	**0**	40	—	—	—	—	0.4	79	0	2.11	5.14	7.23	1.71	103	14.25
Chr18b	1534	1534	37.0	**0**	37	—	—	—	—	0	78.8	0	2.62	0	5.30	0	105	0
Chr20a	2192	2192	149.0	**0**	149	1.82	509	0	331	0	94.6	0.18	3.61	4.38	10.95	0	131	1.82
Chr20b	2298	2298	127.0	**0**	127	5.92	195	2.79	375	5.13	96.4	3.12	3.32	5.40	8.61	0	127	4.96
Chr20c	14142	14142	72.0	**0**	72	0.00	9.23	0	29.49	0	97.8	4.51	1.77	0	13.55	0	140	0
Chr22a	6156	6156	285.0	**0**	285	2.31	201	0	315	0.75	146	0	4.52	0.88	19.11	5.7	164	0.32
Chr22b	6194	6194	283.0	**0**	283	2.58	213	0.97	162	0	152	1.46	5.26	1.68	17.00	8.5	163	0
Chr25a	3796	3796	455.0	**0**	455	2.32	115	0	236	0	194	2.27	5.97	11.17	33.59	0	591	0

Table 7.2b Comparison Results of the WFA with Other Algorithms for Elshafei's, Eschermann's, Hadley's, and Krarup's Instances

Instances	Best known value	Random 2-opt WFA		Best results of WFA		GRASP		ANT		GGA		PGA		IFLS		MSA		PHAS
		Best solution	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}
Els19	17212548	17212548	67	**0**	67	—	—	—	—	0	80.6	0	44.46	—	—	—	—	0
Esc16a	68	68	0.1	**0**	0.1	—	—	—	—	0	47.4	0	5.13	0	3.17	0	61	0
Esc16b	292	292	0.1	**0**	0.1	—	—	—	—	0	48.2	0	0.19	0	2.75	0	60	0
Esc16c	160	160	0.1	**0**	0.1	—	—	—	—	0	53.4	0	0.44	0	4.03	0	68	0
Esc16d	16	16	0.1	**0**	0.1	—	—	—	—	0	53.2	0	0.5	0	3.98	—	—	0
Esc16e	28	28	0.1	**0**	0.1	—	—	—	—	0	46.8	0	0.32	0	2.28	—	—	0
Esc16f	0	0	0.1	**0**	0.1	—	—	—	—	0	46.0	—	—	0	1.11	—	—	0
Esc16g	26	26	0.1	**0**	0.1	—	—	—	—	0	49.8	0	0.29	0	2.77	—	—	0
Esc16h	996	996	0.1	**0**	0.1	—	—	—	—	0	48.0	0	0.22	0	2.13	0	65	0
Esc16i	14	14	0.1	**0**	0.1	—	—	—	—	0	51.6	0	0.16	0	2.05	—	—	0
Esc16j	8	8	0.1	**0**	0.1	—	—	—	—	0	402	0	0.32	0	2.91	—	—	0
Esc32a	130	130	866	**0**	866	1.54	7.03	0	226	0	382	1.52	97.04	0	137	—	—	0
Esc32b	168	168	258	**0**	258	0	2.80	0	40.59	0	400	0	33.61	0	110	—	—	0
Esc32c	642	642	2.6	**0**	2.6	0	0.00	0	0.08	0	389	0	2.01	0	54.7	—	—	0
Esc32d	200	200	39	**0**	39	0	1.92	0	2.13	0	353	0	2.76	0	74.3	—	—	0
Esc32e	2	2	1	**0**	1	0	0.00	0	0.05	0	370	0	0.66	0	46.09	—	—	0
Esc32g	6	6	1	**0**	1	0	0.00	0	0.07	0	371	0	1.27	0	28.41	—	—	0
Esc32h	438	438	141	**0**	141	0	3.41	0	2.64	0	349	0	6.54	0	85.75	—	—	0
Esc64a	116	116	47	**0**	47	—	—	—	—	0	2631	0	194	0	1522	—	—	0
Esc128	64	64	6976	**0**	6976	—	—	—	—	—	—	0	1631	—	—	—	—	0
Had12	1652	1652	1.1	**0**	1.1	—	—	—	—	—	—	0	4.27	0	0.97	0	41	0
Had14	2724	2724	2.7	**0**	2.7	—	—	—	—	—	—	0	10.25	0	1.97	0	64	0
Had16	3720	3720	7.4	**0**	7.4	—	—	—	—	—	—	0	5.38	0.05	3.64	0	88	0
Had18	5358	5358	23	**0**	23	—	—	—	—	—	—	0	18.54	0	6.52	0	118	0
Had20	6922	6922	48	**0**	48	0	2.8	0	159	—	—	0	15.26	0	10.58	0	148	0
Kra30a	88900	88900	2040	**0**	2040	0.00	292	0	199	0	301	0.89	71	1.34	106	—	—	0
Kra30b	91420	91420	2095	**0**	2095	0.32	268	0	140	0	331	0	123	0.13	102	—	—	0.08
Kra32	88700	88700	2052	**0**	2052	—	—	—	—	—	—	—	—	0	172	—	—	0

Table 7.2c Comparison Results of the WFA with Other Algorithms for Li & Pardalos' and Skorin-Kapov's Instances

Instances	Best known value	Random 2-opt WFA		Best results of WFA		GRASP		ANT		GGA		PGA		IFLS		MSA		PHAS
		Best solution	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}
Lipa20a	3683	3683	65	**0**	65	0	0.99	0	107	0	74.8	0	0.59	0	16.11	—	—	0
Lipa20b	27076	27076	76	**0**	76	0	0.66	0	0.00	0	74.4	0	0.39	0	16.78	—	—	0
Lipa30a	13178	13178	352	**0**	352	0	46.43	0	54.85	0	345	0	5.66	0	120	—	—	0.99
Lipa30b	151426	151426	349	**0**	349	0	7.31	0	0.00	0	337	0	2.78	0	122	—	—	0
Lipa40a	31538	31538	3065	**0**	3065	1.13	306.00	1.02	281	0.96	1022	0	19.46	0	490	—	—	—
Lipa40b	476581	476581	1981	**0**	1981	0	6.21	0	0	0	1026	0	9.52	0	486	—	—	—
Lipa50a	62093	62619	4099	0.80	4568	—	—	—	—	0.95	1486	0.82	57.32	1.02	1556	—	—	0
Lipa50b	1210244	1210244	3578	**0**	3578	—	—	—	—	0	1509	0	39.96	0	1462	—	—	0
Lipa60a	107218	108103	6025	0.79	12396	—	—	—	—	0.77	3057	0.64	137	0.84	3668	—	—	0.81
Lipa60b	2520135	2520135	4112	**0**	4112	—	—	—	—	0	3047	0	86.13	0	3724	—	—	0
Lipa70a	169755	170956	8057	0.71	8057	—	—	—	—	0.71	6148	0.62	233	0.77	8067	—	—	—
Lipa70b	4603200	4603200	8503	**0**	8503	—	—	—	—	0	6123	15.9	196	0	7762	—	—	—
Lipa80a	253195	254853	10144	0.63	17685	—	—	—	—	0.61	9519	0.61	373	0.67	15220	—	—	—
Lipa80b	7763962	7763962	10800	**0**	10800	—	—	—	—	0	9499	16.56	332	20.33	15965	—	—	—
Lipa90a	360630	362854	12812	0.57	26123	—	—	—	—	0.58	12358	0.54	592	0.63	27909	—	—	—
Lipa90b	12490441	12490441	25677	**0**	25677	—	—	—	—	0	12319	0	503	0	27788	—	—	—
Sko42	15812	15836	4691	0.03	6961	—	—	—	—	0.25	1006	0.35	365	0.30	614	—	—	0
Sko49	23386	23510	5051	0.32	8158	—	—	—	—	0.21	1252	0.19	714	0.45	1318	—	—	0.05
Sko56	34458	34568	4792	0.20	8537	—	—	—	—	0.02	2976	0.06	907	0.47	2613	—	—	0.12
Sko64	48498	48796	5624	0.31	14405	—	—	—	—	0.22	3788	0.09	1399	0.25	4936	—	—	0
Sko72	66256	66660	7454	0.47	19254	—	—	—	—	0.29	5078	0.21	1987	0.73	8663	—	—	0.03
Sko81	90998	91452	7449	0.50	7449	—	—	—	—	0.20	10964	0.12	2680	0.43	16960	—	—	0.05
Sko90	115534	116922	7640	0.91	21076	—	—	—	—	0.27	12698	0.43	3822	0.45	28787	—	—	0.02
Sko100a	152002	153426	8767	0.94	8767	—	—	—	—	0.21	16608	0.22	1486	1.30	309	—	—	0.19
Sko100b	153890	155288	6724	0.85	7574	—	—	—	—	0.14	14729	0.30	1405	2.34	274	—	—	—
Sko100c	147862	149628	7476	1.15	8576	—	—	—	—	0.20	20314	0.06	873	1.50	284	—	—	—
Sko100d	149576	151196	8335	0.97	19097	—	—	—	—	0.17	20302	0.27	863	1.03	293	—	—	—
Sko100e	149150	151056	11632	0.90	26534	—	—	—	—	0.24	21127	0.33	745	1.55	301	—	—	—
Sko100f	149036	150510	9172	0.91	10200	—	—	—	—	0.29	21479	0.41	781	1.73	285	—	—	—

Table 7.2d Comparison Results of the WFA with Other Algorithms for Nugent's, Roucairol's, Scriabin's, Steinberg's, Thonemann's, and Wilhelm's Instances

Instances	Best known value	Random 2-opt WFA		Best results of WFA		GRASP		ANT		GGA		PGA		IFLS		MSA		PHAS
		Best solution	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}
Nug12	578	578	1.6	**0**	1.6	—	—	—	—	0	19	0	6.84	0	1.41	0	36	0
Nug14	1014	1014	2.0	**0**	2.0	—	—	—	—	—	—	0	7.71	0.39	3.11	—	—	0.2
Nug15	1150	1150	5.8	**0**	5.8	—	—	—	—	0	41.4	0	8.3	0	4.02	0	73	0
Nug16a	1610	1610	16.3	**0**	16.3	—	—	—	—	—	—	0	11.24	0	5.59	—	—	0
Nug16b	1240	1240	18.1	**0**	18.1	—	—	—	—	—	—	0	11.02	0	5.59	—	—	0
Nug17	1732	1732	22.3	**0**	22.3	—	—	—	—	—	—	0	11.95	0	7.31	—	—	0
Nug18	1930	1930	31.1	**0**	31.1	—	—	—	—	—	—	0.31	13.56	0	9.55	—	—	0
Nug20	2570	2570	74.6	**0**	74.6	0	2.53	0	119	0	97.8	0	20.73	0	16.06	0	132	0
Nug21	2438	2438	135	**0**	135	—	—	—	—	—	—	0	29.80	0	20.63	—	—	0
Nug22	3596	3596	119	**0**	119	—	—	—	—	—	—	0	43.82	0	25.84	—	—	0
Nug24	3488	3488	183	**0**	183	—	—	—	—	—	—	0	33.83	0	39.75	—	—	0.06
Nug25	3744	3744	181	**0**	181	—	—	—	—	—	—	0	42.10	0	47.66	—	—	0
Nug27	5234	5234	225	**0**	225	—	—	—	—	—	—	—	—	0	80.56	—	—	0
Nug28	5166	5166	276	**0**	276	—	—	—	—	—	—	—	—	0.12	98.33	—	—	0
Nug30	6124	6124	644	**0**	644	0.42	523	0	181	0.07	354	0.42	109	2.12	117	0.06	887	0.07
Rou12	235528	235528	1.2	**0**	1.2	—	—	—	—	0	19.6	0	0.30	0	1.06	0	35	0
Rou15	354210	354210	3.4	**0**	3.4	—	—	—	—	0	34.6	0	0.56	0	2.95	0.71	71	0
Rou20	725522	725522	57.8	**0**	57.8	0	165	0	245	0.16	75.2	0	1.43	0.02	11.73	0.06	127	0
Scr12	31410	31410	1.3	**0**	1.3	—	—	—	—	0	18.8	0	0.44	0	1.11	0	38	0
Scr15	51140	51140	4.6	**0**	4.6	—	—	—	—	0	35.2	0	0.42	0	3.09	0	78	0
Scr20	110030	110030	23.4	**0**	23.4	0	157	0	46.1	0	79.6	0	1.57	0	12.69	2.13	137	0
Ste36a	9526	9526	3057	**0**	3057	1.81	276	0.76	295	0.27	710	0	221	0	204	—	—	—
Ste36b	15852	15852	3425	**0**	3425	0.92	180	0.25	213	—	—	0	235	3.43	222	—	—	—
Ste36c	8239110	8239110	3696	**0**	3696	0.89	142	0.33	321	—	—	0	24.07	—	—	—	—	—
Tho30	149936	149936	1379	**0**	1379	0.00	216	0	288	0	396	0	132	0.29	119	—	—	—
Tho40	240516	240620	3704	0.04	3704	1.17	184	0.66	312	0.32	958	0.05	344	0.53	502	—	—	—
Tho150	8133398	8238058	21600	1.29	21600	—	—	—	—	—	—	0.41	729	—	—	—	—	—
Wil50	48816	48916	3933	0.06	11856	—	—	—	—	0.07	2115	0	695	0.28	1499	—	—	—
Wil100	273038	274446	15320	0.34	23725	—	—	—	—	0.2	20544	0.15	1252	0.27	51121	—	—	—

Table 7.2e Comparison Results of the WFA with Other Algorithms for Taillard's Instances

Instances	Best known value	Random 2-opt WFA		Best results of WFA		GRASP		ANT		GGA		PGA		IFLS		MSA		PHAS
		Best solution	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}	Time (s)	Δ_{Best}
Tai12a	224416	224416	0.7	**0**	0.7	—	—	—	—	—	—	0	0.18	0	1.05	0	35	0
Tai12b	39464925	39464925	0.5	**0**	0.5	—	—	—	—	—	—	0	0.62	0	1.03	0.03	53	0
Tai15a	388214	388214	8.1	**0**	8.1	—	—	—	—	—	—	0	0.69	0	2.99	0.39	73	0.01
Tai15b	51765268	51765268	4.2	**0**	4.2	—	—	—	—	—	—	0	0.7	0	3.05	0.47	77	0
Tai17a	491812	491812	18	**0**	18	—	—	—	—	—	—	0.55	0.96	0.38	5.55	0	86	0.56
Tai20a	703482	703482	860	**0**	860	0	484	0	160	—	—	0.84	1.38	0.47	11.42	0.21	124	0.8
Tai20b	122455319	122455319	22	**0**	22	—	—	—	—	—	—	0	2.70	0	12.52	5.6	167	0
Tai25a	1167256	1169030	1157	**0**	7268	1.43	355	0.55	206	—	—	0.77	3.27	2.00	33.03	—	—	1.57
Tai25b	344355646	344355646	332	**0**	332	—	—	—	—	—	—	0	3.77	5.59	35	—	—	0
Tai30a	1818146	1832590	1969	0.57	2623	1.58	265	1.46	332	—	—	1.34	6.72	1.11	83.06	—	—	1.37
Tai30b	637117113	637117113	3169	**0**	3169	—	—	—	—	—	—	0	14.11	2.22	81	—	—	0
Tai35a	2422002	2436540	2477	0.59	6185	1.90	531	1.64	232	—	—	1.29	12.09	1.24	177	—	—	1.3
Tai35b	283315445	283315445	4749	**0**	4749	—	—	—	—	—	—	0	23.9	3.54	186	—	—	0.19
Tai40a	3139370	3160484	4612	0.67	4612	2.20	325	2.05	253	—	—	1.08	18.37	1.85	354	—	—	1.7
Tai40b	637250948	637250948	5355	**0**	5355	—	—	—	—	—	—	0	36.95	5.60	328	—	—	0
Tai50a	4938796	5031472	5354	1.46	11342	—	—	—	—	—	—	1.31	58.21	2.25	1104	—	—	2.48
Tai50b	458821517	458926056	7166	0.02	7166	—	—	—	—	—	—	0	64.77	0.42	1032	—	—	0
Tai60a	7205962	7342990	9962	1.55	14450	—	—	—	—	—	—	1.79	104	2.75	2740	—	—	2.37
Tai60b	608215054	612153786	8584	0.65	8584	—	—	—	—	—	—	0	148	0.47	2621	—	—	0.02
Tai64c	1855928	1855928	5834	**0**	5834	—	—	—	—	—	—	0	28.96	0.03	237	—	—	0
Tai80a	13511780	13821180	10084	1.87	27967	—	—	—	—	—	—	1.41	360	2.46	11333	—	—	2.37
Tai80b	818415043	827982667	15800	1.17	15800	—	—	—	—	—	—	0.03	424	2.79	10533	—	—	0
Tai100a	21052466	21538854	11274	1.76	36544	—	—	—	—	—	—	1.29	785	2.33	35781	—	—	—
Tai100b	1185996137	1198498100	20268	1.05	20268	—	—	—	—	—	—	0.32	855	0.52	34336	—	—	—
Tai150b	498896643	508566248	44677	1.94	44677	—	—	—	—	—	—	0.2	3414	0.38	290186	—	—	—
Tai256c	44759294	44896638	35434	0.27	367250	—	—	—	—	—	—	0.16	1956	0.27	73180	—	—	—

Table 7.3 Improved Results of the WFA Variants for the QAP Instances Not Optimally Solved by 2-Opt WFA

Instances	Best known value	Random 2-opt WFA		Systematic generator 2-opt WFA		Random 2-opt mirror WFA		WFA-3-opt	
		Best solution	Time (s)	Best solution	Time (s)	Best solution	Time (s)	Best solution	Time (s)
Lipa50a	62093	62619	4099	62703	5099	62666	6481	***62593***	***4568***
Lipa60a	107218	108103	6025	108172	10534	***108070***	***12396***	108070	12396
Lipa80a	253195	254853	10144	254840	17053	***254800***	***17685***	254800	17685
Lipa90a	360630	362854	12812	362906	19905	***362673***	***26123***	362673	26123
Sko42	15812	15836	4691	***15816***	***6961***	15830	8252	15816	6961
Sko49	23386	23510	5051	***23460***	***8158***	23474	8971	23460	8158
Sko56	34458	34568	4792	***34528***	***8537***	34558	11047	34528	8537
Sko64	48498	48796	5624	***48648***	***14405***	48758	13885	48648	14405
Sko72	66256	66660	7454	***66570***	***19254***	66820	15072	66570	19254
Sko90	115534	116922	7640	116968	18859	116632	20212	***116590***	***21076***
Sko100b	153890	155288	6724	155728	18431	155398	21747	***155204***	***7574***
Sko100c	147862	149628	7476	149862	17480	149990	19819	***149564***	***8576***
Sko100d	149576	151196	8335	151186	19884	***151022***	***19097***	151022	19097
Sko100e	149150	151056	11632	151140	19147	150588	25371	***150498***	***26534***
Sko100f	149036	150510	9172	151032	18680	150794	21531	***150390***	***10200***
Tai25a	1167256	1169030	1157	1169030	2453	***1167256***	***7268***	1167256	7268
Tai30a	1818146	1832590	1969	***1828576***	***2623***	1830918	10287	1828576	2623
Tai35a	2422002	2436540	2477	***2436458***	***6185***	2443826	9234	2436458	6185
Tai50a	4938796	5031472	5354	***5010798***	***11342***	5026322	15100	5010798	11342
Tai60a	7205962	7342990	9962	***7317694***	***14450***	7353798	19163	7317694	14450
Tai80a	13511780	13821180	10084	13790286	12708	***13764720***	***27967***	13764720	27967
Tai100a	21052466	21538854	11274	21577638	17538	***21422344***	***36544***	21422344	36544
Tai256c	44759294	44896638	35434	***44879868***	***367250***	44881948	103680	44879868	367250
Wil50	48816	48916	3933	48856	13800	***48846***	***11856***	48846	11856
Wil100	273038	274446	15320	274244	18825	274608	27522	***273980***	***23725***

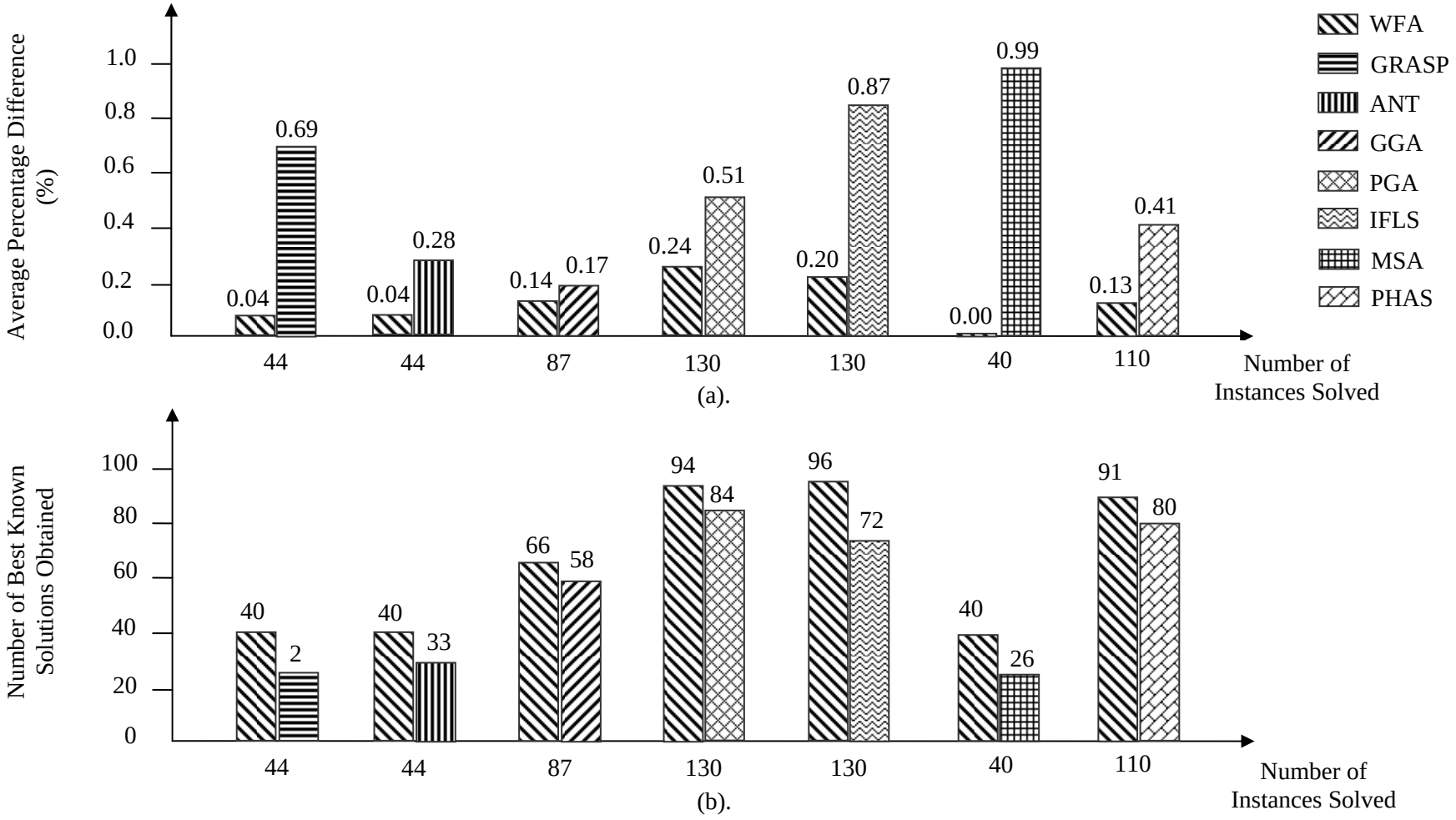


Figure 7.3 Comparison of WFA with Other Algorithms on (a). Average Percentage Difference; and (b). Number of Best Known Solutions Obtained

When compared with other meta-heuristic algorithms, namely GRASP, ANT, GGA, PGA, IFLS, MSA, and PHAS, it can be seen from Tables 7.2a to 7.2e that WFA outperforms GRASP, ANT, and MSA in all the instances, and also outperforms GGA,PGA, IFLS, and PHAS in many of the instances. When comparing the overall performance using the average relative percentage difference, as well as the number of the best known solutions obtained, the WFA dominates these meta-heuristic algorithms as shown in Figure 7.3, especially when compared with the GRASP, the ANT, the IFLS, and the MSA. This shows that the WFA is able to obtain good results when compared to other efficient meta-heuristic algorithms.

7.2 **Vehicle Routing Problem**

In this section, we present the two-level WFA for solving VRP. In this algorithm, the first level focuses on solving the VRP with relaxation of integrality constraints. Then at the second level, a modified WFA uses the initial solutions obtained by the first level to search for the optimal solutions. Here, we illustrate the performance of the 2LWFA with the CVRP. The performance of the 2LWFA has been tested on some CVRP benchmark instances obtained from the literature. The experimental results obtained are compared with the best known solutions found from the literature, and they demonstrate the efficiency of the 2LWFA for solving the CVRP.

7.2.1 Capacitated Vehicle Routing Problem

In the field of logistics and supply chain management, how to arrange an appropriate supplier-to-customer assignment and determine an efficient distribution routing is very

important. A good solution can not only improve the efficiency in operation, but also significantly reduce operating costs. Many problems that originate from these concerns are classified as VRPs, which are NP-hard problems (Toth and Vigo, 2002).

The CVRP can be considered as one of the typical VRPs. This problem consists of a fleet of vehicles with pre-specified limited capacity, which has to serve a set of customers with specific demands. In this thesis, we have focused on the CVRP with one depot. Hence, the vehicles must start and end at the same depot. The CVRP aims to assign the vehicles to the customers and to find the efficient routes of the vehicles so that total travel distance (or time) is minimized. The problem has some important constraints, such as (1) total demand of the customers served by a vehicle should not exceed the capacity of the vehicle, (2) one customer can only be served by one vehicle, (3) all customers must be served, (4) the number of vehicles used cannot exceed the number of given vehicles. In this problem, the distance between the depot and customers, as well as between a customer and other customers, are also given.

There are many research works related to the CVRP. Among them, the works that formulate the CVRP as an integer linear programming problem have generally led to effective exact solution approaches for this problem. In this research work, we have used the integer linear programming formulation presented in Kulkarni and Bhave (1985) and corrected in Imdat et al. (2004). The integrality constraints of the formulation are relaxed to allow the first level of the 2LWFA to find good initial solutions.

7.2.2 Two-level WFA for the CVRP

A 2LWFA has been proposed for solving the CVRP. The first level of this algorithm is to solve the mathematical programming model of the CVRP with the relaxation of the integrality constraints. This is to reduce the computation time of obtaining good initial solutions for the 2LWFA. Then, the obtained optimal solution is adjusted by a check-and-fit procedure, and a perturbation scheme is applied to generate a set of initial feasible solutions for the second level of this algorithm. At the second level, a modified WFA is applied to search for optimal solutions from the initial seed solutions.

7.2.2.1 First Level

The efficiency and effectiveness of solving the CVRP based on the mathematical programming formulation depends on the solvers used. Most solvers require large amounts of computation time and may fail to return solutions for problems with large size. To avoid this problem and reduce the computation effort required, we relax the integrality constraints of the CVRP. In particular, we focus on the decision variable X_{ij} in the mathematical programming formulation of the CVRP. It is a binary variable that is equal to 1 if and only if customers (or depot) i and j are connected. We relax these variables to real numbers in [0, 1] and solve the resulting relaxed model with the commercial optimization package LINGO 5.0. This relaxed CVRP model can be described by the equations from (7.15) to (7.23). In these equations, n is the number of customers, m is the number of routes, and C_{ij} represents the cost or distance between customers i and j. Variables X_{ij} or X_{ji} exist only if $q_i + q_j \le Q$, where q_i and q_j represent the demand of customers i and j respectively, and Q is the maximum capacity of the vehicle. A variable

u_i is associated with each customer i, which is used to formulate equation (7.20) to ensure that the solution contains no sub-tours disconnected from the depot. Other equations represent the common constraints of the CVRP described in Section 7.2.1.

$$\text{Min} \quad \sum_{i \neq j} C_{ij} X_{ij} \tag{7.15}$$

$$\text{S.t.:} \quad \sum_{j=1}^{n} X_{0j} = m, \tag{7.16}$$

$$\sum_{i=1}^{n} X_{i0} = m, \tag{7.17}$$

$$\sum_{\substack{j=0 \\ j \neq i}}^{n} X_{ij} = 1 \quad \text{for} \quad i = 1, ..., n, \tag{7.18}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} X_{ij} = 1 \quad \text{for} \quad j = 1, ..., n, \tag{7.19}$$

$$u_i - u_j + QX_{ij} \leq Q - q_j \quad \text{for} \quad i, j = 1, ..., n \ \text{ and } \ i \neq j, \tag{7.20}$$

$$q_i \leq u_i \leq Q \quad \text{for} \quad i = 1, ..., n, \tag{7.21}$$

$$X_{ij} \in [0;1] \quad \text{for} \quad i, j = 0, ..., n \ \text{ and } \ i \neq j, \tag{7.22}$$

$$m \geq 1 \ \text{ and } \ m \in Z^+, \tag{7.23}$$

For the resulting optimal solution, we arrange the values of decision variables in a descending order. Those X_{ij} variables with large values can be considered to be potential connections of customers (or depot) i and j. As such, we will assign these X_{ij} to be 1. Also, we check the feasibility of the assignment based on the constraints of the CVRP, and fit in the most appropriate assignment. This check-and-fit procedure is performed iteratively until a feasible solution of the CVRP is determined. Then, a perturbation scheme based on the cross-exchange procedure in Taillard et al. (1997) is used to generate a set of seed solutions from the feasible solution. This cross-exchange procedure can be illustrated by

Figure 7.4. In this figure, the square and circles represent the depot and the customers respectively. Each route begins and ends at the same depot (or square). Firstly, we remove two edges (A1, A1') and (B1, B1') from route 1, and two edges (A2, A2') and (B2, B2') from route 2. Next, the segments (A1', B1) and (A2', B2) which may include some customers are swapped to formulate the new routes, i.e., new route 1 contains the new edges (A1, A2') and (B2, B1'), while new route 2 contains (A2, A1') and (B1, B2'). These new routes are accepted only when they satisfy all constraints of the CVRP.

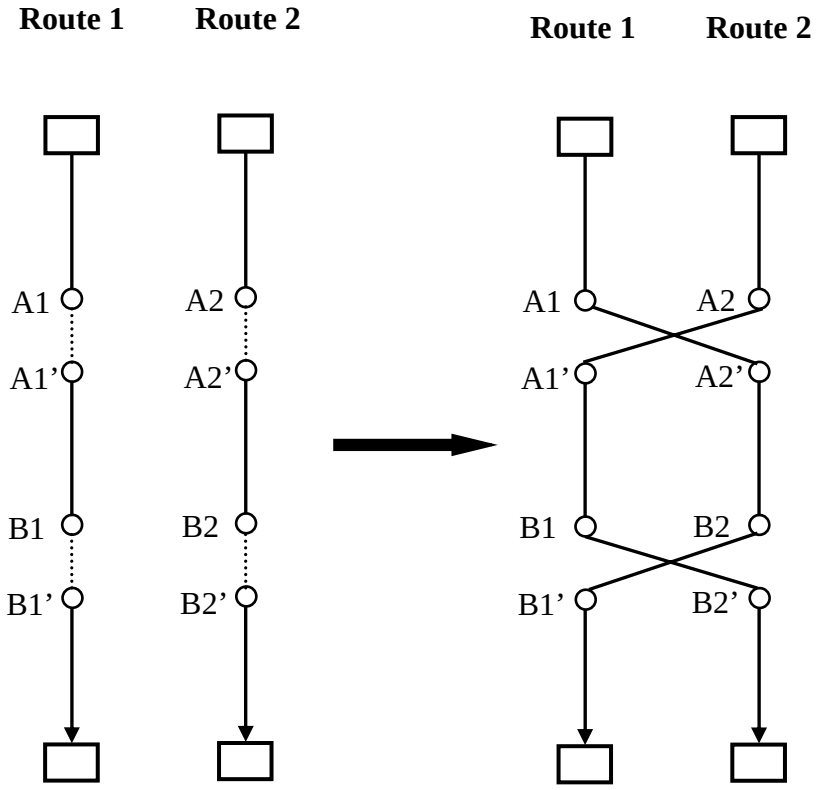


Figure 7.4 The Cross-Exchange Procedure (Taillard et al., 1997)

We now describe the check-and-fit procedure in greater detail and illustrate it with a benchmark instance of Christofides and Eilon (1969) for the CVRP. The instance used is

E-n13-k4, which has 12 customers served by 4 vehicles. Firstly, we use LINGO 5.0 to solve the problem, and the values of the decision variables obtained are as follows: $X_{0,1}$ = $X_{0,2}$ = $X_{0,6}$ = $X_{0,9}$ = $X_{1,0}$ = $X_{2,0}$ = $X_{3,0}$ = $X_{6,0}$ = $X_{9,12}$ = $X_{12,3}$ = 1; $X_{4,7}$ = $X_{4,11}$ = $X_{7,8}$ = $X_{5,10}$ = $X_{7,4}$ = $X_{7,10}$ = $X_{8,5}$ = $X_{8,11}$ = $X_{10,5}$ = $X_{10,7}$ = $X_{11,4}$ = $X_{11,8}$ = 0.5; $X_{2,7}$ = 0.28; and all the rest of the variables $X_{i,j}$ = 0. Here, index 0 denotes the depot. Secondly, we apply the check-and-fit procedure to obtain a feasible solution. In particular, we consider the decision variables whose value is 1. The corresponding customers in the decision variables are assigned in the same route. For example, we have $X_{0,1}$ = $X_{1,0}$ = 1, and also we assign customer 1 into route 1. Similarly, we obtain 4 partial routes, i.e., route 1: (1), route 2: (2), route 3: (6), and route 4: (9, 12, 3). Route 4 consists of 3 customers since $X_{0,9}$ = $X_{9,12}$ = $X_{12,3}$ = $X_{3,0}$ = 1. Next, we consider whether any customer is connected with a customer assigned in the partial routes. In this case, we see that only $X_{2,7}$ = 0.28, and thus we assign customer 7 into route 2. In addition, since $X_{7,4}$ = $X_{4,11}$ = 0.5, we also assign customers 4 and 11 into route 2, i.e., route 2: (2, 7, 4, 11). Although we still have $X_{11,8}$ = 0.5, we do not assign customer 8 into route 2, since including customer 8 would violate the constraint of maximum capacity of each vehicle. Moreover, we chose the customer 4 instead of customer 8 to assign into route 2 after assigning customer 7. This is because we choose based on the minimum distance between customers. Here, the distance between customers 7 and 4 is shorter than the distance between customers 7 and 8. Since we do not have any nonzero decision variable that may connect the unassigned customers with the last customer in the partial routes, we consider and choose the customer with minimum distance between it and the last customer in partial routes. Here, we will assign customer 10 into route 3. Because of $X_{10,5}$ = 0.5, we also assign customer 5 right after customer 10 in route 3. After applying the check-and-fit procedure, we construct a feasible solution with 4 complete routes, i.e., route

1: (1, 8), route 2: (2, 7, 4, 11), route 3: (6, 10, 5), and route 4: (9, 12, 3). The total corresponding travel distance is 307. Based on the cross-exchange procedure described above, the feasible solution is used to generate a set of seed solutions that are then used as initial solutions in the modified WFA.

A flow chart of the first level of the 2LWFA for the CVRP is shown in Figure 7.5.

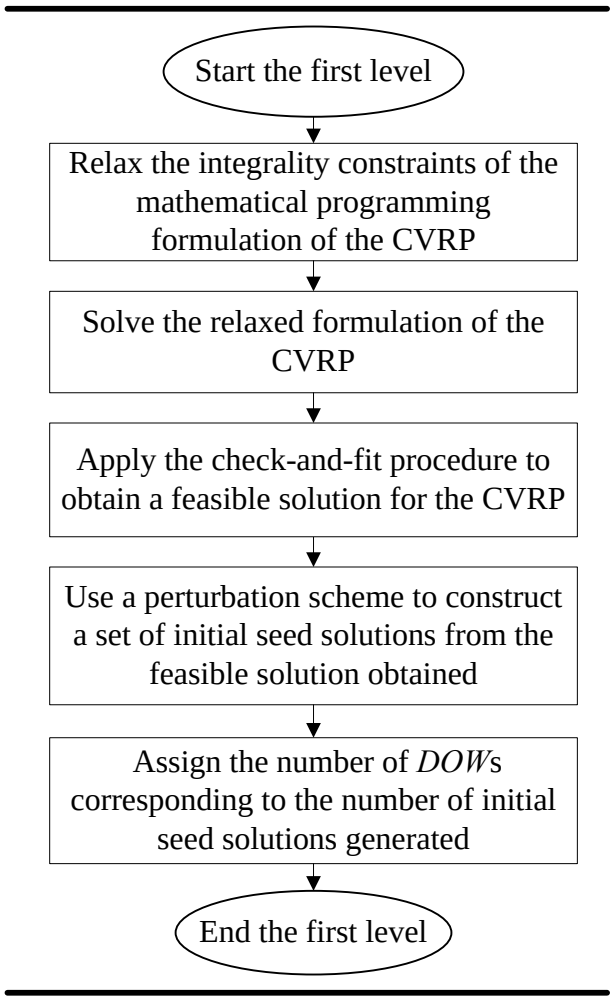


Figure 7.5 Flow Chart of the First Level of the 2LWFA for the CVRP

7.2.2.2 Second Level

After constructing a set of seed solutions, a modified WFA is applied to search for optimal solutions from the seed solutions. Basically, this algorithm is similar to the WFA described in Chapter 3. However, a new solution representation for the CVRP is being used in the modified WFA. Also, a variable neighborhood structure based on the *k*-opt algorithm is used in the erosion process to enhance the flexibility and efficiency of this process. From the literature, the *k*-opt algorithm with $k \geq 5$ has obtained solutions with insignificant improved quality compared to the *k*-opt algorithm with other smaller values of *k*, and yet requires a large amount of computation time. Hence, we chose the value of *k* to be 3 and 4 for the variable neighborhood structure in the erosion process of the modified WFA. The implementation of this variable neighborhood search is similar to the method in Hansen and Mladenovic (2001). For the neighborhood structure in the exploration phase of the modified WFA, the 2-opt algorithm is used.

The WFA encodes a feasible solution of an optimization problem and its objective value into a *DOW*, which is a component of a cloud representing a pool of solutions. For the CVRP, we consider the values of the coordinates (X, Y) as the longitude and latitude in the position of *DOW* on the ground, while the total traveling distance/time is encoded as the altitude. In some instances when the values of coordinates (X, Y) for the depot and customers are not known, we use the solution representation of the CVRP with specific routes for vehicles. Actually, this solution can also be expressed by the values of coordinates (X, Y) with the corresponding sequences.

We use an example to illustrate the solution representation of the CVRP. Here, we denote integer 0 to be the depot, and other integers from {1, 2,…, *n*} to be the customers. For example, a solution representation may then be shown as [0, 2, 1, 3, 0, 5, 4, 6]. This solution includes two vehicles: a vehicle starts from the depot to serve the customers with routing 2, 1, and 3, and goes back to the depot; another vehicle starts from the depot to serve the customers with routing 5, 4, and 6, and goes back to the depot. With this solution representation, integer 0 can appear many times in the solution depending on the number of vehicles given, but other integers can only appear once in the solution. This is suitable for the constraints of the VRP presented in this chapter. Figure 7.6 shows an example of a *DOW* and its positional vector components for the CVRP with n = 6 customers and 2 vehicles.

Solution to VRPs	**0**	2	1	3	**0**	5	4	6	***DOW***
Coordinate X	145	159	151	130	145	163	128	146	Longitude
Coordinate Y	215	261	264	254	215	247	252	246	Latitude
Objective value	Total travel distance = 244								Altitude

Figure 7.6 An Example of Solution Representation in the 2LWFA for the CVRP

A flow chart of the modified WFA of 2LWFA for the CVRP is shown in Figure 7.7.

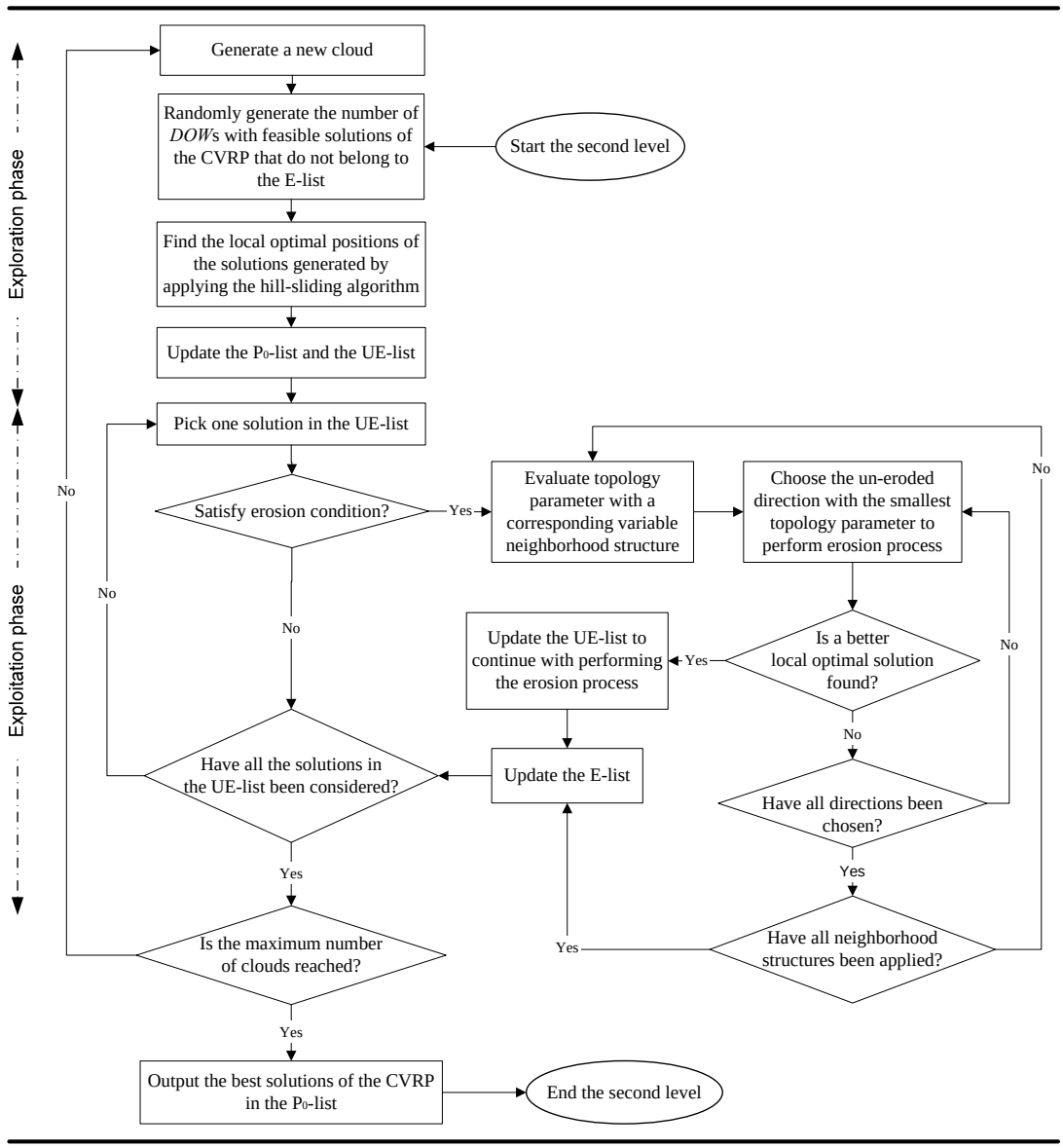


Figure 7.7 Flow Chart of the Modified WFA for the CVRP

7.2.3 Preliminary Experiments

The 2LWFA has been coded using Visual Basic 6.0 and linked with LINGO 5.0. All preliminary experiments have been performed on an Intel Centrino Duo 1.60 GHz CPU with 1.5 GB of RAM. Here, the computational complexity of the 2LWFA for the CVRP is determined based on the neighborhood structure used and the erosion process of this

algorithm. In particular, the 2LWFA used *k*-opt neighborhood structure (k = 3 or 4), and the worst possibility of the erosion process is to find for all n directions. Thus, the computational complexity of the 2LWFA is estimated to be $O(n^5)$.

These experiments are carried out on the benchmark instances of Christofides and Eilon (1969) and Fisher (1994) taken from the literature for the CVRP. The best known values of the benchmark instances from the literature are also used as reference values for the evaluation of the WFA for solving the CVRP. These values may be obtained from the optimal solutions of the benchmark instances used, or the best solutions found by some algorithm so far. The results obtained are shown in Table 7.4. The choice of parameters for the 2LWFA was determined by design-of-experiment methods and the best values of the parameters include *MaxCloud* = 20, *MaxPop* = 10, *MinEro* = 3, and *MaxUIE* = 5.

From Table 7.4, we see that the 2LWFA can obtain optimal solutions for the instances with small and medium size within reasonable computation time. For instances with larger size, i.e., F-n72-k4, E-n76-k7, and E-n101-k8, the proposed algorithm can also find solutions with a relative deviation of 1.7%, 0.6%, and 4.4% over the optimal/best known solution, respectively. The average relative percentage difference for all instances is only 0.6%.

The results from Table 7.4 also show that the solution obtained by 2LWFA achieved an average improvement of 20.97% over the initial solution obtained by applying the check-and-fit procedure to the solution found using LINGO. We find that these improvements are more significant for the instances with larger size.

Table 7.4 Experimental Results for the CVRP

Instance	Optimal/Best known value	Initial solution after applying check-and-fit procedure	2LWFA	CPU time (secs)	
				Lingo	WFA
E-n13-k4	247	307	247	0.2	0.50
E-n22-k4	375	390	375	0.5	1.75
E-n23-k3	569	623	569	0.6	4.47
E-n30-k3	534	582	534	1.0	7.50
E-n33-k4	835	910	835	1.5	35.25
E-n51-k5	521	627	521	3.0	120.15
E-n76-k7	683	843	687	18.0	1261.72
E-n101-k8	817	1012	853	50.0	4840.14
F-n45-k4	724	839	724	2.6	102.66
F-n72-k4	237	425	241	20.5	1435.72

7.3 **Conclusions**

In this chapter, the WFA has been developed for solving other combinatorial optimization problems, such as the QAP and the VRP. To solve the QAP by the proposed algorithm with enhanced solution diversification and intensification capabilities, a systematic *DOW* generator scheme to distribute the positions of *DOW* is applied, while neighborhood structures, such as the 2-opt mirror and 3-opt, are used to focus on strong searching of

promising regions. The benchmark problem sets from the QAPLIB (Burkard et al., 1997) are used to evaluate the performance of the WFA. The computational results show that the WFA is able to generate optimal solutions for many benchmark problems of QAP, and near-optimal solutions for the remaining problems. The proposed algorithm is also compared with other meta-heuristic algorithms from the literature. The results of the comparison show that the WFA compares favorably with other meta-heuristic algorithms used to solve the QAP. These results have been reported in Ng and Tran (2011).

To solve the CVRP efficiently, we developed a two-level WFA. The first level of the proposed algorithm is to solve the mathematical programming model of the CVRP with the relaxation of the integrality constraints. At the second level, a modified WFA is applied to search for optimal solutions from the initial solutions obtained from the first level. The results of the preliminary experimentations show the potential of the 2LWFA to solve the CVRP, as well as other types of VRPs. These results have been reported in Tran and Ng (2011[b]).

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH WORK

In this thesis, we have introduced a novel nature-inspired algorithm, known as the water flow algorithm, for solving combinatorial optimization problems. The proposed algorithm has simulated the hydrological cycle in meteorology and the erosion phenomenon in nature, which represent the solution exploration and exploitation capabilities of the algorithm, respectively. Many characteristics of water flow in nature, such as water always moving to lower positions, distributing onto many places on the ground, and eroding terrain, are imitated to design the operators of this algorithm. The WFA has been applied to solve a variety of NP-hard combinatorial optimization problems, i.e., flow shop scheduling problem, flexible flow shop scheduling problem with intermediate buffers, quadratic assignment problem, and capacitated vehicle routing problem. Also, this algorithm has been developed to solve multi-objective scheduling problem, which is an NP-hard optimization problem with many practical applications in modern production environment.

According to the literature review, there are limited meta-heuristic algorithms inspired by the behaviors of water flow in nature. These works have not demonstrated the efficiency of the algorithms for solving various optimization problems. Hence, with the success of the WFAs for solving such single-objective and multi-objective problems as illustrated in this thesis, we have put forth a new and effective nature-inspired optimization approach. This approach may be applied to solve other optimization problems by adjusting the components of the algorithms appropriately.

Chapter 8 is organized as follows. In Section 8.1, we present some conclusions of the thesis. The contributions of this thesis are also highlighted in this section. Section 8.2 provides some possible future research works.

8.1 Conclusions

This thesis focuses on constructing the general WFA and the implementation of this algorithm for solving NP-hard combinatorial optimization problems, such as the flow shop scheduling problem, flexible flow shop scheduling problem with intermediate buffers, quadratic assignment problem, and vehicle routing problem. Both single-objective and multi-objective optimization approaches of the WFA were also developed to solve such optimization problems. From the experimental results, we can conclude that the WFA is a promising method that is able to obtain good quality solutions to the optimization problems within reasonable computation time.

The main contributions of this thesis consist of six parts, and they can be outlined as follows:

(1). A novel nature-inspired algorithm, known as the water flow algorithm, for solving NP-hard optimization problems was constructed. The WFA has mostly simulated the characteristics of water flow in nature and the components of the hydrological cycle in meteorology. This algorithm consists of two major phases, the solution exploration and exploitation phases, which are inspired by the hydrological cycle and the erosion phenomenon, respectively. The WFA has achieved a balance between the solution exploration and exploitation capabilities to search for optimal solutions in reasonable computation time. In the algorithm, the number of controlled parameters defined by users is small. Hence, the computation time needed by this algorithm is significantly reduced, which helps to increase the performance of the WFAs.

(2). With some modifications, the WFA could be developed to solve several single-objective NP-hard optimization problems, such as the permutation flow shop scheduling problem, flexible flow shop scheduling problem with intermediate buffers, quadratic assignment problem, and capacitated vehicle routing problem. For the PFSP, we have used a basic version of the WFA to solve this problem. The algorithm obtained the best known solutions for almost all the benchmark instances used, and a new best known solution of a Heller benchmark instance was found by this algorithm. In addition, the WFA outperforms several meta-heuristic algorithms used in the computational comparisons.

(3). For the FFSP with intermediate buffers, some components of the WFA were modified to solve this scheduling problem. In particular, an improved procedure for constructing a complete schedule of the FFSP was integrated into the algorithm.

The procedure helps the WFA to ignore the number of machines at each stage, which decreases the computational complexity of the algorithm for this problem. Moreover, we combined the amount of precipitation and its falling force to form a flexible erosion capability in this algorithm. This helps the erosion process to focus on exploiting promising regions strongly. The experimental results show that the WFA is an efficient algorithm for solving benchmark instances from the literature and randomly generated instances. Many improved solutions to the benchmark problems were found by this algorithm. Also, the results demonstrate the potential of the algorithm to solve real-world problems, such as in maltose syrup production. Moreover, the comparison results show that the WFA outperforms other meta-heuristic algorithms, such as the tabu search and memetic algorithm, for solving the FFSP with intermediate buffers.

(4). The WFA is able to solve the QAP effectively. In this version of the WFA, a systematic *DOW* generator scheme to distribute the positions of *DOW*s was proposed to increase the exploration capability of the algorithm, while the neighborhood structures, such as the 2-opt mirror and 3-opt, were integrated to focus on strong searching of promising regions. The WFA obtained the best known solutions for 99 out of the 134 instances from the QAPLIB within reasonable computation time. The average relative percentage difference of the algorithm for all the 134 instances was found to be 0.20%. In addition, the WFA outperforms all the algorithms when compared in terms of average percentage difference and number of the best known solutions obtained.

(5). A two-level WFA was developed to solve vehicle routing problems. The first level of this algorithm is to solve the mathematical programming model of the VRPs with the relaxation of the integrality constraints. At the second level, a modified WFA is then applied to search for optimal solutions from the initial solutions obtained from the first level. In the modified WFA, a variable neighborhood structure based on the *k*-opt algorithm was used to enhance the flexibility and efficiency of the erosion process. In this thesis we illustrated the performance of the 2LWFA for solving the capacitated vehicle routing problem. The experimental results show the potential of the 2LWFA to solve this problem as well as other types of VRPs.

(6). In addition to the capability of solving single-objective optimization problems, the WFA is also able to solve multi-objective optimization problems by modifying and integrating some specialized components. In this thesis, the WFA was developed to solve the multi-objective FFSP with intermediate buffers. In this algorithm, landscape analysis was performed to determine the weights of objective functions, which guide *DOW*s to exploit potential regions and move towards the optimal Pareto solution set. Also, the evaporation and precipitation processes were included into this algorithm to enhance the solution exploitation capability of the algorithm in potential neighboring regions. Moreover, an improvement process for reinforcing the final Pareto solution set obtained was proposed. The experimental results, based on benchmark instances taken from the literature and randomly generated instances, demonstrate the effectiveness and efficiency of the MOWFA.

The comparison results also show that the MOWFA outperforms other algorithms for the test instances.

8.2 Future Research Work

In this section, we present some possible future research work and directions for the WFA. These consist of improvement and application of the algorithms.

Firstly, the choice of parameters for the WFAs can be improved. While the current choice of parameters used in the WFAs for single-objective and multi-objective optimization problems has obtained good computational performance, further research on how to improve the choice of parameters used in the WFAs can still be performed. This would lead to greater efficiency for the WFAs when solving the optimization problems, as well as the possibility of obtaining solutions with better quality by the algorithms.

Secondly, when we use the erosion process based on the lower bound of the optimization problem, i.e. for the flexible flow shop scheduling problem with intermediate buffers, the performance of this process depends on the quality of lower bound used. In this thesis, we only used the lower bounds from the literature. Hence, obtaining a good lower bound for the scheduling problem to be used in the WFA may be considered as possible future research work.

Thirdly, we have illustrated the potential of the WFAs for solving other optimization problems in this thesis. The WFA and the MOWFA can be used to solve other types of single-objective and multi-objective optimization problems respectively, that are similar to the optimization problems investigated in this thesis, by adapting some components of the

algorithms appropriately. Depending on the specific structure of optimization problems considered, we can customize the WFAs to solve the problems efficiently and this would certainly be part of future research work. Some examples of such potential optimization problems include the job shop scheduling problem and traveling salesman problem.

In addition, we can apply the WFA to solve the scheduling problems with other important objective functions, such as minimizing total flow time of jobs, minimax tardiness, or minimizing total idle time of machines. This helps to allow a more comprehensive evaluation of the performance of the WFA. We can even develop the WFA for solving scheduling problems with data noise and uncertainty which are commonly found in most practical single-objective and multi- objective problems. Stochastic techniques for dealing with such problems can then be integrated into the WFA.

Lastly, we can extend the WFA for solving continuous optimization problems by designing appropriate neighborhood structures for continuous variables. Furthermore, some smoothing functions or methods in the field of continuous optimization may be integrated into the erosion process of the WFA to enhance the performance of the algorithm.

REFERENCES

[1]. Abbiw-Jackson R., B. Golden, S. Raghavan, and E. Wasil. A Divide-and-Conquer Local Search Heuristic for Data Visualization, Computers and Operations Research, *33*(11), pp. 3070–3087. 2006.

[2]. Abraham A., C. Grosan, and V. Ramos. Stigmergic Optimization. Netherlands: Springer-Verlag Berlin Heidelberg. 2006.

[3]. Agarwal A., S. Colak, and E. Eryarsoy. Improvement Heuristic for the Flow Shop Scheduling Problem: An Adaptive Learning Approach, European Journal of Operational Research, *169*(3), pp. 801–815. 2006.

[4]. Ahuja R.K., T.L. Magnanti, and J.B. Orlin. Network Flows: Theory, Algorithms, and Applications. New Jersey, US: Prentice Hall. 1993.

[5]. Ahuja R.K., J.B. Orlin, and A. Tiwari. A Greedy Genetic Algorithm for The Quadratic Assignment Problem, Computers and Operations Research, *27*(10), pp.917–934. 2000.

[6]. Akturk M.S, and M.B. Yildirim. A New Lower Bounding Scheme for the Total Weighted Tardiness Problem, Computers and Operations Research, *25*(4), pp. 265–278. 1998.

[7]. Applegate D., R.E. Bixby, V. Chvatal, and W. Cook. Finding Cuts in the TSP. A Preliminary Report Distributed at the Mathematical Programming Symposium, Ann Arbor, Michigan. 1994.

[8]. Arkin E.M., R. Hassin, and M. Sviridenko. Approximating the Maximum Quadratic Assignment Problem, Information Processing Letters, *77*(1), pp. 13–16. 2001.

[9]. Azizoglu M., E. Cakmak, and S. Kondakci. A Flexible Flow Shop Problem with Total Flow Time Minimization, European Journal of Operational Research, *132*(3), pp. 528–538. 2001.

[10]. Banks A., J. Vincent, and C. Anyakoha. A Review of Particle Swarm Optimization - Part I: Background and Development, Natural Computation, *6*(4), pp.467–484. 2007.

[11]. Basak A., S. Pal, S. Das, A. Abraham, and V. Snasel. A Modified Invasive Weed Optimization Algorithm for Time-Modulated Linear Antenna Array Synthesis, IEEE Congress on Evolutionary Computation (CEC'10). 2010.

[12]. Basturk B., and D. Karabogo. An Artificial Bee Colony (ABC) Algorithm for Numerical Function Optimization. In: IEEE Swarm Intelligence Symposium 2006, May 12-24, Indianapolis, IN, USA. 2006.

[13]. Basu S., C. Chaudhuri, M. Kundu, M. Nasipuri, and D.K. Basu. Text Line Extraction from Multi-Skewed Handwritten Documents, Pattern Recognition, *40*(6), pp. 1825–1839. 2007.

[14]. Bitam S., M. Batouche, and E. Talbi. A Survey on Bee Colony Algorithms. In: Proceedings of IEEE International Symposium on Parallel & Distributed Processing (IPDPSW' 10), pp. 1–8. 2010.

[15]. Blanchard A., S. Elloumi, A. Faye, and N. Wicker. A Cutting Algorithm for the Quadratic Assignment Problem, INFOR, *41*(1), pp. 35–49. 2003.

[16]. Bokhari S.H. Assignment Problems in Parallel and Distributed Computing. Kluwer Academic Publishers, Boston, MA. 1987.

[17]. Brodic D., and Z. Milivojevic. Reference Text Line Identification Based on Water Flow Algorithm. In: Proceedings of ICEST 2009, SP-2 Sect, Veliko Tarnovo, Bulgaria. 2009[a].

[18]. Brodic D., and Z. Milivojevic. Modified Water Flow Method for Reference Text Line Detection. In: Proceedings of ICCS 2009, Sofia, Bulgaria. 2009[b].

[19]. Brodic D., and Z. Milivojevic. An Approach to Modification of Water Flow Algorithm for Segmentation and Text Parameters Extraction. In: Proceedings of DoCEIS 2010, IFIP AICT 314, pp. 324–331. 2010.

[20]. Buffa E.S., G.C. Armour, and T.E. Vollmann. Allocating Facilities with CRAFT, Harvard Business Review, *42*(2), pp. 136–158. 1964.

[21]. Burkard R.E., E. Cela, G. Rote, and G.J. Woeginger. The Quadratic Assignment Problem with A Monotone Anti-Monge and A Symmetric Toeplitz Matrix: Easy and Hard Cases. In: Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization, Vancouver, British Columbia, Canada, pp. 204–218. 1996.

[22]. Burkard R.E., S.E. Karisch, and F. Rendl. QAPLIB - A Quadratic Assignment Problem Library, Journal of Global Optimization, *10*(4), pp. 391–403. 1997.

[23]. Cela E. The Quadratic Assignment Problem: Theory and Algorithms. In: D.Z. Du, P. Pardalos (Eds.), Combinatorial Optimization, Kluwer Academic Publishers, London. 1998.

[24]. Chatterjee S., C. Carrera, and L. Lynch. Genetic Algorithms and Traveling Salesman Problems, European Journal of Operational Research, *93*(3), pp. 490–510. 1996.

[25]. Christofides N., and S. Eilon. An Algorithm for the Vehicle Dispatching Problem, Operational Research Quarterly, *20*(3), pp. 309–318. 1969.

[26]. Christofides N., A. Mingozzi, P. Toth, and C. Sandi. Combinatorial Optimization. John Wiley & Sons. 1979.

[27]. Dantzig G.B., and J.H. Ramser. The Truck Dispatching Problem, Management Science, *6*(1), pp. 80–91. 1959.

[28]. Darwin C. On the Origin of Species by Means of Natural Selection. John Murray, London. 1859.

[29]. Dawkins R. The Selfish Gene. Oxford: Oxford University Press. 1976.

[30]. Demidenko V.M., G. Finke, and V.S. Gordon. Well Solvable Cases of the Quadratic Assignment Problem with Monotone and Bimonotone Matrices, Journal of Mathematical Modeling and Algorithms, *5*(2), pp. 167–187. 2006.

[31]. Deneubourg J.L., S. Goss, J.M. Pasteels, D. Fresneau, and J.P. Lachaud. Self-Organization Mechanisms in Ant Societies (II): Learning in Foraging and Division of Labor. In: From Individual to Collective Behavior in Social Insects (J.M. Pasteels and J.L. Deneubourg, Eds.), Experientia Supplementum, 54, pp. 177–196. 1987.

[32]. Dorigo M. Optimization, Learning and Natural Algorithms. Ph.D. thesis, Politecnico di Milano, Italie. 1992.

[33]. Dorigo M., and L.M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Transaction on Evolutionary Computation, *1*(1), pp.53–66. 1997.

[34]. Dorigo M., G.D. Caro, and L.M. Gambardella. Ant Algorithms for Discrete Optimization, Artificial Life, *5*(2), pp.137–172. 1999.

[35]. Duman E., and I. Or. The Quadratic Assignment Problem in the Context of the Printed Circuit Board Assembly Process, Computers and Operations Research, *34*(1), pp. 163–179. 2007.

[36]. Eschermann B., and H.J. Wunderlich. Optimized Synthesis of Self-Testable Finite State Machines. In: Proceedings of the 20th International Symposium Fault-Tolerant Computing (FTCS 20), Newcastle Upon Tyne, UK, pp. 390–397. 1990.

[37]. Eusuff M.M., and K.E. Lansey. Optimization of Water Distribution Network Design Using the Shuffled Frog Leaping Algorithm, Journal of Water Resources Planning and Management, *129*(3), pp. 210–25. 2003.

[38]. Farooq M. From the Wisdom of the Hive to Intelligent Routing in Telecommunication Networks: A Step towards Intelligent Network Management through Natural Engineering, PhD Thesis, University of Dortmund, Germany. 2006.

[39]. Gasimov R.N., and O. Ustun. Solving the Quadratic Assignment Problem Using F-MSG Algorithm, Journal of Industrial and Management Optimization, *3*(2), pp. 173–191. 2007.

[40]. Goldberg D.E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley. 1989.

[41]. Goudie A. The Nature of the Environment. Oxford: Blackwell Science. 1993.

[42]. Grabowski J., and J. Pempera. Sequencing of Jobs in Some Production System, European Journal of Operational Research, *125*(3), pp. 535–550. 2000.

[43]. Grosan C., and A. Abraham. Stigmergic Optimization: Inspiration, Technologies and Perspectives, Studies in Computational Intelligence (SCI), *31*, pp. 1–24. 2006.

[44]. Gutin G., and A. Yeo. Polynomial Approximation Algorithms for TSP and QAP with A Factorial Domination Number, Discrete Applied Mathematics, *119*(1–2), pp. 107–116. 2002.

[45]. Haddad O.B., A. Afshar, and M.A. Marino. Honey Bees Mating Optimization Algorithm (HBMO): A New Heuristic Approach for Engineering Optimization. In:

Proceedings of the First International Conference on Modeling, Simulation and Applied Optimization, Sharjah, UAE. 2005.

[46]. Hahn P.M., W.L. Hightower, T.A. Johnson, M. Guignard-Spielberg, and C. Roucairol. Tree Elaboration Strategies in Branch and Bound Algorithms for Solving the Quadratic Assignment Problem, Yugoslavian Journal of Operational Research, *11*(1), pp. 41–60. 2001.

[47]. Hansen P., and N. Mladenovic. Variable Neighborhood Search: Principles and Applications, European Journal of Operational Research, *130*(3), pp. 449–467. 2001.

[48]. Ho J.C., and Y.L. Chang. A New Heuristic for the *N*-Job, *M*-Machine Flow Shop Problem, European Journal of Operational Research, *52*(2), pp. 194–202. 1991.

[49]. Holland J.H. Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press. 1975.

[50]. Holy M. Erosion and Environment. Elmsford: Pergamon. 1982.

[51]. Hubert L.J. Assignment Methods in Combinatorial Data Analysis. Marcel Dekker Inc., New York. 1987.

[52]. Hull P. Glucose Syrups: Technology and Applications. UK, Wiley-Blackwell. 2010.

[53]. Huynh T.H. A Modified Shuffled Frog Leaping Algorithm for Optimal Tuning of Multivariable PID Controllers, IEEE International Conference on Industrial Technology (ICIT'08), Chengdu, China, pp. 1–6. 2008.

[54]. Imdat K., L. Gilbert, and B. Tolga. A Note on the Lifted Miller–Tucker–Zemlin Tour Elimination Constraints for the Capacitated Vehicle Routing Problem, European Journal of Operational Research, *158*(3), pp. 793–795. 2004.

[55]. Ishibuchi H., T. Yoshida, and T. Murata. Balance between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling, IEEE Transactions on Evolutionary Computation, *7*(2), pp. 204–223. 2003.

[56]. Johnson S.M. Optimal Two-And Three-Stage Production Schedules with Setup Times Included, Naval Research Logistics Quarterly, *1*(1), pp. 61–68. 1954.

[57]. Karimkashi S., and A.A. Kishk. Invasive Weed Optimization and Its Features in Electromagnetics, IEEE Transactions on Antennas and Propagation, *58*(4), pp. 1269–1278. 2010.

[58]. Kennedy J., and R.C Eberhart. Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, pp. 1942–1948. 1995.

[59]. Kim I.K., D.W. Jung, and R.H. Park. Document Image Binarization Based on Topographic Analysis Using A Water Flow Model, Pattern Recognition, *35*(1), pp. 265–277. 2002.

[60]. Knowles J.D., and D.W. Corne. On Metrics for Comparing Nondominated Sets. In: Proceedings of Congress on Evolutionary Computation, pp. 711–716. 2002.

[61]. Koulamas C. A New Constructive Heuristic for Flow Shop Scheduling Problem, European Journal of Operational Research, *105*(1), pp. 66–71. 1998.

[62]. Koopmans T., and M.J. Beckmann. Assignment Problems and the Location of Economics Activities, Econometric, *25*(1), pp. 53–76. 1957.

[63]. Kulkarni R.V., and P.R. Bhave. Integer Programming Formulations of Vehicle Routing Problems, European Journal of Operational Research, *20*(1), pp. 58–67. 1985.

[64]. Lei D. Multi-Objective Production Scheduling: A Survey, International Journal of Advance Manufacturing Technology, *43*(9–10), pp. 926–938. 2009.

[65]. Li Y., P.M. Pardalos, and M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. In: P.M. Pardalos, H. Wolkowicz (Eds.), Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 16, pp. 237–261. 1994.

[66]. Li W.T., X.W. Shi, Y.Q. Hei, S.F. Liu, and J. Zhu. A Hybrid Optimization Algorithm and Its Application for Conformal Array Pattern Synthesis, IEEE Transactions on Antennas and Propagation, *58*(10), pp. 3401–3406. 2010.

[67]. Liao C.J., C.T. Tseng, and P. Luarn. A Discrete Version of Particle Swarm Optimization for Flow Shop Scheduling Problems, Computers and Operations Research, *34*(10), pp. 3099–3111. 2007.

[68]. Lim M., Y. Yuan, and S. Omatu. Extensive Testing of A Hybrid Genetic Algorithm for Solving Quadratic Assignment Problems, Computational Optimization and Applications, *23*(1), pp. 47–64. 2002.

[69]. Liong S.Y., and M.D. Atiquzzaman. Optimal Design of Water Distribution Network Using Shuffled Complex Evolution, Journal of the Institution of Engineers, Singapore, *44*(1), pp. 93–107. 2004.

[70]. Lukasik S., and S. Zak. Firefly Algorithm for Continuous Constrained Optimization Task, ICCCI 2009, Lecture Notes in Artificial Intelligence (Eds. N.T. Nguyen, R. Kowalczyk, S.M. Chen), 5796, pp. 97–100. 2009.

[71]. Mallahzadeh A.R., H. Oraizi, and Z. Davoodi-Rad. Application of the Invasive Weed Optimization Technique for Antenna Configuration, Progress in Electromagnetics Research, *79*, pp. 137–150. 2008.

[72]. Maniezzo V., and A. Colorni. The Ant System Applied to the Quadratic Assignment Problem, IEEE Transactions on Knowledge and Data Engineering, *11*(5), pp. 769–778. 1999.

[73]. Martello S., and P. Toth. Knapsack Problems: Algorithms and Computer Implementations. New York: John Wiley and Sons. 1990.

[74]. McCormick S.T., M.L. Pinedo, S. Shenker, and B. Wolf. Sequencing in an Assembly Line with Blocking to Minimize Cycle Time, Operations Research, *37*(6), pp. 925–936. 1989.

[75]. Mehrabian A.R., and C. Lucas. A Novel Numerical Optimization Algorithm Inspired from Weed Colonization, Ecological Informatics, *1*(4), pp. 355–366. 2006.

[76]. Mehrabian A.R., and A.Y. Koma. Optimal Positioning of Piezoelectric Actuators on A Smart Fin Using Bio-Inspired Algorithms, Aerospace science and technology, *11*(2–3), pp.174–182. 2007.

[77]. Merz P., and B. Freisleben. A Genetic Local Search Approach to the Quadratic Assignment Problem. In: Proceedings of the 7th International Conference on Genetic Algorithms, San Diego, CA: Morgan Kaufmann, pp. 465–72. 1997.

[78]. Merz P., and B. Freisleben. Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem, IEEE Transactions on Evolutionary Computation, *4*(4), pp. 337–352. 2000.

[79]. Michael P. Fundamentals of Physical Geography (2nd Edition). PhysicalGeography.net. Retrieved 2007-03-19. 2006.

[80]. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer Verlag, Berlin. 1992.

[81]. Moscato P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report 826, California Institute of Technology, Pasadena, California, USA. 1989.

[82]. Moscato P., and C. Cotta. A Gentle Introduction to Memetic Algorithms. In: Handbook of Metaheuristics (F. Glover and G. Kochenberger, eds.), pp. 105–144. Kluwer Academic Publishers, Boston MA. 2003.

[83]. Moscato P., and M.G. Norman. A Memetic Approach for the Traveling Salesman Problem—Implementation of A Computational Ecology for Combinatorial Optimization on Message-Passing Systems. In: International Conference on Parallel Computing and Transputer Application, Amsterdam, Holland: IOS Press, pp. 177–86. 1992.

[84]. Mostafa S.A., R. Mahnaz, R.K. Ashkan, and C. Lucas. A Study of Electricity Market Dynamics Using Invasive Weed Colonization Optimization, IEEE Symposium on Computational Intelligence and Games (CIG'08). 2008.

[85]. Nagano M.S., R. Ruiz, and L.A.N. Lorena. A Constructive Genetic Algorithm for Permutation Flowshop Scheduling, Computers and Industrial Engineering, *55*(1), pp. 195–207. 2008.

[86]. Nakrani S., and C. Tovey. On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers, Adaptive Behaviors, *12*(3–4), pp. 223–240. 2004.

[87]. Nawaz M., E.E. Enscore Jr, and I. Ham. A Heuristic Algorithm for the M-Machine, N-Job Flow-Shop Sequencing Problem, OMEGA: International Journal of Management Science, *11*(1), pp. 91–95. 1983.

[88]. Ng K.M., and T.H. Tran. A Water Flow Algorithm for Solving the Quadratic Assignment Problem, Submitted to Applied Mathematics and Computation. 2011.

[89]. Oh H.H., K.T. Lim, and S.I. Chien. An Improved Binarization Algorithm Based on A Water Flow Model for Document Image with Inhomogeneous Backgrounds, Pattern Recognition, *38*(12), pp. 2612–2625. 2005.

[90]. Onwubolu G., and D. Davendra. Scheduling Flow Shops Using Differential Evolution Algorithm, European Journal of Operational Research, *171*(2), pp. 674–692. 2006.

[91]. Osman I.H., and C.N. Potts. Simulated Annealing for Permutation Flow Shop Scheduling, OMEGA International Journal of Management Science, *17*(6), pp. 551–557. 1989.

[92]. Otsu N. A Threshold Selection Method from Gray-Level Histograms, IEEE Transaction on Systems, Man, and Cybernetic, *9*(1), pp. 62–66. 1979.

[93]. Papadimitriou C.H., and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Englewood Cliffs, N.J.: Prentice Hall. 1982.

[94]. Pedersen S., and H. Vang-Hendriksen. Method for Production of Maltose and/or Enzymatically Modified Starch. World Intellectual Property Organization, International Publication Number: WO 01/16349 A1. 2001.

[95]. Pinedo M. Scheduling: Theory, Algorithms, and Systems. Upper Saddle, N.J.: Prentice Hall. 2002.

[96]. Pinedo M. Planning and Scheduling in Manufacturing and Services. New York: Springer. 2005.

[97]. Qahri Saremi H., B. Abedin, and A. Meimand Kermani. Website Structure Improvement: Quadratic Assignment Problem Approach and Ant Colony Metaheuristic Technique, Applied Mathematics and Computation, *195*(1), pp. 285–298. 2008.

[98]. Qian B., L. Wang, D.X. Huang, W.L. Wang, and X. Wang. An Effective Hybrid De-Based Algorithm for Multi-Objective Flow Shop Scheduling with Limited Buffers, Computers and Operations Research, *36*(1), pp. 209–233. 2009.

[99]. Quadt D., and H. Kuhn. A Taxonomy of Flexible Flow Line Scheduling Procedures, European Journal of Operational Research, *178*(3), pp. 686–698. 2007.

[100]. Rajendran C., and H. Ziegler. Ant-Colony Algorithms for Permutation Flowshop Scheduling to Minimize Makespan/Total Flowtime of Jobs, European Journal of Operational Research, *155*(2), pp. 426–438. 2004.

[101]. Rameshkumar K., R.K. Suresh, and K.M. Mohanasundaram. Discrete Particle Swarm Optimization (DPSO) Algorithm for Permutation Flow Shop Scheduling to

Minimize Makespan. Lecture Notes in Computer Science, 3612, pp. 572–581. 2005.

[102]. Ramkumar A.S., S.G. Ponnambalam, and N. Jawahar. Iterated Fast Local Search Algorithm for Solving Quadratic Assignment Problems, Robotics and Computer-Integrated Manufacturing, *24*(3), pp. 392–401. 2008.

[103]. Ramkumar A.S., S.G. Ponnambalam, and N. Jawahar. A Population-Based Hybrid Ant System for Quadratic Assignment Formulations in Facility Layout Design, International Journal of Advanced Manufacturing Technology, *44*(5–6), pp. 548–558. 2009.

[104]. Rashidi E., M. Jahandar, and M. Zandieh. An Improved Hybrid Multi-Objective Parallel Genetic Algorithm for Hybrid Flow Shop Scheduling with Unrelated Parallel Machines, International Journal of Advanced Manufacturing Technology, *49*(9–12), pp. 1129–1139. 2010.

[105]. Ravindran D., A. Noorul Haq, S.J. Selvakuar, and R. Sivaraman. Flow Shop Scheduling with Multiple Objective of Minimizing Makespan and Total Flow Time, International Journal of Advanced Manufacturing Technology, *25*(9–10), pp. 1007–1012. 2005.

[106]. Reeves C.R. A Genetic Algorithm for Flowshop Sequencing, Computers and Operations Research, *22*(1), pp.5–13. 1995.

[107]. Resnick M. Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. Bradford Books. 1997.

[108]. Ribas I., R. Leisten, and J.M. Framinan. Review and Classification of Hybrid Flow Shop Scheduling Problems from a Production System and a Solutions Procedure Perspective, Computers and Operations Research, *37*(8), pp. 1439–1454. 2010.

[109]. Richardson P. Bats. Natural History Museum, London. 2008.

[110]. Rinnooy Kan A.H.G. Machine Scheduling Problems: Classification, Complexity and Computations. The Hague: Nijhoff. 1976.

[111]. Rossin D.F., M.C. Springer, and B.D. Klein. New Complexity Measures for the Facility Layout Problem: An Empirical Study Using Traditional and Neural Network Analysis, Computers and Industrial Engineering, *36*(3), pp. 585–602. 1999.

[112]. Roth M., and S. Wicker. Termite: A Swarm Intelligent Routing Algorithm for Mobile Wireless Ad-Hoc Networks, Studies in Computational Intelligence (SCI), *31*, pp. 155–184. 2006.

[113]. Ruiz R., and J.A. Vazquez-Rodriguez. The Hybrid Flow Shop Scheduling Problem, European Journal of Operational Research, *205*(1), pp. 1–18. 2010.

[114]. Sahni S., and T. Gonzalez. P-complete Approximation Problems, Journal of the ACM, *23*(3), pp. 555–565. 1976.

[115]. Sawik T. Mixed Integer Programming for Scheduling Flexible Flow Lines with Limited Intermediate Buffers, Mathematical and Computer Modeling, *31*(13), pp.39–52. 2000.

[116]. Sawik T. Mixed Integer Programming for Scheduling Surface Mount Technology Lines, International Journal of Production Research, *39*(14), pp. 3219–3235. 2001.

[117]. Sayadi M.K., R. Ramezanian, and N. Ghaffari-Nasab. A Discrete Firefly Meta-Heuristic with Local Search for Makespan Minimization in Permutation Flow Shop Scheduling Problems. International Journal of Industrial Engineering Computations, *1*, pp. 1–10. 2010.

[118]. Seeley T.D., S. Camazine, and J. Sneyd. Collective Decision Making in Honey Bees: How Colonies Choose Among Nectar Sources, Behavioural Ecoloy and Sociobiology, *28*(4), pp. 277–290. 1991.

[119]. Sepehri-Rad H., and C. Lucas. A Recommender System Based on Invasive Weed Optimization Algorithm. In: IEEE Congress on Evolutionary Computation (CEC 2007), pp. 4297–4304. 2007.

[120]. Shah-Hosseini H. Problem Solving By Intelligent Water Drops. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2007), pp. 3226–3231. 2007.

[121]. Shah-Hosseini H. Intelligent Water Drops Algorithm: A New Optimization Method for Solving the Multiple Knapsack Problem, International Journal of Intelligent Computing and Cybernetics, *1*(2), pp. 193–212. 2008.

[122]. Shah-Hosseini H. The Intelligent Water Drops Algorithm: A Nature-Inspired Swarm-Based Optimization Algorithm, International Journal of Bio-Inspired Computation, 1(1–2), pp. 71–79. 2009.

[123]. Sherali H.D., S.C. Sarin, and M.S. Kodialam. Models and Algorithms for a Two-Stage Production Process, Production Planning and Control, *1*(1), pp. 27–39. 1990.

[124]. Silva J.D.L., E.K. Burke, and S. Petrovic. An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling. In: Lecture Notes in Economics and Mathematical Systems, *535*, pp. 91–129. 2004.

[125]. Singh S.P., and R.R.K. Sharma. Two-Level Modified Simulated Annealing Based Approach for Solving Facility Layout Problem, International Journal of Production Research, *46*(13), pp. 3563–3582. 2008.

[126]. Suliman S.M.A. A Two-Phase Heuristic Approach to the Permutation Flow-Shop Scheduling Problem, International Journal of Production Economics, *64*(1–3), pp. 143–152. 2000.

[127]. Taillard E. Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem, European Journal of Operational Research, *47*(1), pp. 65–74. 1990.

[128]. Taillard E. Comparison of Iterative Searches for the Quadratic Assignment Problem, Location Science, *3*(2), pp. 87–105. 1995.

[129]. Taillard E., P. Badeau, M. Gendreau, F. Guertin, and J.Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows, Transportation Science, *31*(2), pp. 170–186. 1997.

[130]. Tang L.X., and H. Xuan. Lagrangian Relaxation Algorithms for Real-Time Hybrid Flowshop Scheduling with Finite Intermediate Buffers, Journal of the Operational Research Society, *57*(3), pp. 316–324. 2006.

[131]. Tantar E., C. Dhaenens, J.R. Figueira, and E. Talbi. A Priori Landscape Analysis in Guiding Interactive Multi-Objective Metaheuristics. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2008), pp. 4104–4111. 2008.

[132]. Tasgetiren M.F., Y.C. Liang, M. Sevkli, and G. Gencyilmaz. A Particle Swarm Optimization Algorithm for Makespan and Total Flowtime Minimization in the Permutation Flowshop Sequencing Problem, European Journal of Operational Research, *177*(3), pp. 1930–1947. 2007.

[133]. Tavakkoli-Moghaddam R., N. Safaei, and F. Sassani. A Memetic Algorithm for the Flexible Flow Line Scheduling Problem with Processor Blocking, Computers and Operations Research, *36*(2), pp. 402–414. 2009.

[134]. Teodorovic D., T. Davidovic, and M. Selmic. Bee Colony Optimization: The Applications Survey, ACM Transactions on Computational Logic, pp. 1–20. 2011.

[135]. Toth P., and D. Vigo. Models, Relaxations and Exact Approaches for the Capacitated Vehicle Routing Problem, Discrete Applied Mathematics, *123*(1–3), pp. 487–512. 2002.

[136]. Tran T.H., and K.M. Ng. A Water Flow Algorithm for Flexible Flow Shop Scheduling with Limited Intermediate Buffers. In: Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'09), Dublin, Ireland, pp. 606–616. 2009.

[137]. Tran T.H., and K.M. Ng. A Water-Flow Algorithm for Flexible Flow Shop Scheduling with Intermediate Buffers, Journal of Scheduling, In Press. 2010.

[138]. Tran T.H., and K.M. Ng. A Water Flow Algorithm for Multi-Objective Flexible Flow Shop Scheduling with Intermediate Buffers, Submitted to Computers and Operations Research. 2011[a].

[139]. Tran T.H., and K.M. Ng. Two-Level Water-Flow Algorithm for Vehicle Routing Problems, Submitted to the 5th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'11), Phoenix, Arizona, USA. 2011[b].

[140]. Van Veldhuizen D.A. Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations. PhD dissertation, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, US. 1999.

[141]. Wang W., B. Wu, Y. Zhao, and D. Feng. Particle Swarm Optimization for Open Vehicle Routing Problem, Lecture Notes in Computer Science, *4114*, pp. 999–1007. 2006.

[142]. Wardono B. Algorithms for the Multi-Stage Parallel Machine Problem with Buffer Constraints. Ph.D. Dissertation, North Carolina State University. 2001.

[143]. Wardono B., and Y. Fathi. A Tabu Search Algorithm for the Multi-Stage Parallel Machine Problem with Limited Buffer Capacities, European Journal of Operational Research, *155*(2), pp. 380–401. 2004.

[144]. Wei Z., X.F. Xu, and S.C. Deng. Evolutionary Algorithm for Solving Multi-Objective Flow Shop Scheduling Problem, Computers Integrated Manufacturing Systems, *12*, pp. 1227–1234 (in Chinese). 2006.

[145]. Widmer M., and A. Hertz. A New Heuristic Method for the Flow Shop Sequencing Problem, European Journal of Operational Research, *41*(2), pp. 186–193. 1989.

[146]. Wittrock R.J. An Adaptable Scheduling Algorithm for Flexible Flow Lines, Operations Research, *36*(3), pp. 445–453. 1988.

[147]. Wong K.Y., and P.C. See. A Hybrid Ant Colony Optimization Algorithm for Solving Facility Layout Problems Formulated as Quadratic Assignment Problems, Engineering Computations: International Journal for Computer-Aided Engineering and Software, *27*(1), pp. 117–128. 2010.

[148]. Wu T.H., S.H. Chung, and C.C. Chang. A Water Flow-Like Algorithm for Manufacturing Cell Formation Problems, European Journal of Operational Research, *205*(2), pp. 346–360. 2010.

[149]. Yang F.C., and Y.P. Wang. Water Flow-Like Algorithm for Object Grouping Problems, Journal of the Chinese of Industrial Engineers, *24*(6), pp. 475–488. 2007.

[150]. Yang X., Q. Lu, C. Li, and X. Liao. Biological Computation of the Solution to the Quadratic Assignment Problem, Applied Mathematics and Computation, *200*(1), pp. 369–377. 2008.

[151]. Yang X.S. Engineering Optimization via Nature-Inspired Virtual Bee Algorithms, IWINAC 2005, Lecture Notes in Computer Science, *3562*, pp. 317–323. 2005.

[152]. Yang X.S. Nature-Inspired Metaheuristic Algorithms. UK: Luniver. 2008.

[153]. Yang X.S. A New Metaheuristic Bat-Inspired Algorithm. In: Nature Inspired Cooperative Strategies for Optimization (Eds. J.R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin, *284*, Springer, pp. 65-74. 2010.

[154]. Yu G. Industrial Applications of Combinatorial Optimization. Boston: Kluwer Academic Publishers. 1998.

[155]. Zhang X., Y. Wang, G. Cui, Y. Niu, and J. Xu. Application of A Novel IWO to the Design of Encoding Sequences for DNA Computing, Computers and Mathematics with Applications, *57*(11-12), pp. 2001–2008. 2009.

[156]. Zhi X., X. Xing, Q. Wang, L. Zhang, X. Yang, C. Zhou, and Y. Liang. A Discrete PSO Method for Generalized TSP Problem. In: Proceedings of International Conference on Machine Learning and Cybernetics, pp. 2378–2383. 2004.