New Methods for Solving the Minimum Weighted Latency Problem

by

Ziqi Wei

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In this thesis, I study the methods to solve an NP-hard problem, minimum weighted latency problem (MWLP). The well-studied NP-hard problem minimum latency problem (MLP) can be seen as a special case of MLWP. I introduce the motivation of studying MWLP at the beginning. After describing the existing methods, I propose a Mixed Integer Programming algorithm for MWLP. As for the main body of this thesis, I study the heuristic algorithms for MWLP. I use five different classic heuristics to test the effectiveness of heuristic when solving MWLP. After that, two new meta-heuristics are proposed and tested. Both of them produce better results compared to the classic ones. At last, I describe two real-world applications for this problem. MWLP model is tested and proved effective for both applications.

Preface

Part of chapter 3 of this thesis has been published as Z.Wei and M.H.MacGregor, Heuristic Approaches for Minimum Weighted Latency Problem, Icmmita 2016, vol. 71, pp. 552C556, 2017. I was responsible for the algorithm design and experiments as well as the manuscript composition. M.H. MacGregor was the supervisory author and was involved with concept formation and manuscript composition.

Acknowledgement

I dedicate this thesis to my parents. Without them, I could not and would not accomplish it. I would also like to thank my supervisor Dr. Mike MacGregor. He is such a wise and kind person. He helped me on both my academic work and my daily life. Dr. Zack Friggstad helped me on my work in graph theory. We had many very inspiring discussions. My roommate Wanxin Gao and I had many great time during the years we spent together. I will miss the summer nights we drunk and chatted in our balcony. I have to thank my friends Baoliang Wang, Jundong Li, Zhaoxing Bu, and Ping Jin. You guys made me feel like home in the alien land. The last person I would like to thank is my Cat. The happiness you brought me has deeply affected and changed me.

I have to thank the Chinese Scholarship Council. They financially supported me.

At last, I would like to thank Canada. She is such a beautiful country. During the past six years, I met so many nice people in my daily life. At here, I realize the power of tenderness and kindness for the first time in my life.

Contents

List of Tables

List of Figures

1 Introduction

In this first chapter, I describe the motivation for studying MWLP in my thesis. After the motivation, I will introduce the background and the research status of MWLP. The problem's significance and its difficulty will be described in the third section. At the end of this chapter, I describe the organization of this thesis in detail.

1.1 Motivation

The travelling salesman problem (TSP) is a well-known NP-hard problem[1]. It is also a very well-studied problem[2, 3, 4]. The objective of its optimization is to find a Hamiltonian tour that minimizes the mobile agent's entire travelling path and visits every vertex in the map. The problem is very significant in both theoretical computer science and operations research communities. In addition to these theoretical contributions, the original formulation of TSP can be used directly in real-world applications such as very large-scale integrated (VLSI) circuit design, logistics design, and multi-task planning problems. With slight modification, it can be used in other application areas, such as DNA sequencing and astronomy. However, even though it is a widely studied and powerful formulated problem, it still has some shortcomings when dealing with particular situations. For example, if a tsunami occurs suddenly, the rescue group should arrive at every village as quickly as possible. The requirement for a good design for the rescue team's travelling route is proposed. The main idea is to minimize the sum of all the villagers' waiting time. TSP is not suitable for this problem.

The reason that TSP is not able to solve this issue is that TSP is designed as a server-oriented problem. It focuses on the requirement of the mobile agent but not the vertices (customers). However, the emergency relief problem described above is customer-oriented. Owing to the shortcomings of TSP, researchers have designed a new formulation called the minimum latency problem (MLP)[1, 5]. Similar to TSP, the MLP attempts to find a path that visits every vertex in the map. However, its optimization objective is to minimize the sum of every vertex's waiting time (latency). By using the purest formulation of MLP, it can solve the emergency relief problem. The customer-oriented problem is prevalent in the real world. In particular, in modern highly developed economies, satisfying the customers' requirements is essential. Nevertheless, the model of MLP has its shortcomings. In the emergency relief setting, every village suffers significant losses. Every human life is priceless, so the importance of saving every life and minimizing losses is immeasurable. If we consider this problem from the other side, the importance of rescuing each village can be seen as equal. However, in other situations, the importance of different customers can vary.

Suppose a cargo truck driver Bob is planning to transport a vehicle of various species of fish to some restaurants from the harbour. His customers, the restaurants, have different preferences. Some of them may like the high-quality species, such as largemouth bass. Others may be interested in some common species, for example, fathead minnows. Bob must deliver all the species of fish his customers require if he does not want to lose his job. To simplify the problem, we assume the restaurants will accept all the fish Bob delivers to them as long as the fish are alive and they receive certain requested species of fish. In addition, the fish die at a constant rate. In Bob's dream, his truck is equipped with refrigeration equipment. All the fish will survive till the time they arrive at the restaurant that requested them. However, in reality, he only has a regular truck without any kind of equipment to keep the fish alive. Now the question is how can Bob minimize his loss during the transportation? What mathematical model should he use to achieve his goal?

Obviously, the question Bob is facing is a customer-oriented problem. However, MLP is unable to deal with this circumstance. Different customers have different requirements. Here, the requirements are different species of fish. They have different values. Bob's loss is a function of two variables, both the fish prices and the restaurants' waiting time. MLP contains only one variable, the restaurants' waiting time, which is not sufficient in this situation. Here, we introduce a new mathematical model, MWLP.

MWLP contains all the features of MLP. MLP can be seen as a particular case of MWLP where weights of vertices are all 1. From the customers' side, it tries to satisfy the customers' requirements and minimize their waiting time. There is no doubt in this case that MWLP is a customer-oriented problem. MWLP may also classify the customers by the price of the species of fish they request, which minimizes the driver's loss. This feature makes MWLP a server-oriented problem. That is, MWLP can be seen as both customer-oriented and server-oriented. This attractive feature allows MWLP to be applied to a wider range of applications. MWLP can be implemented to help corporations to increase their profits, cargo drivers to improve their customer satisfaction, and even the disaster relief teams to save more lives. More details of MWLP's applications are given in Chapter 5.

To the best of the author's knowledge, no systematic research on MWLP has been presented previously. I propose an exact algorithm based on integer programming and two meta-heuristic algorithms for this problem in this thesis. A comparison of the capacity of different classic heuristics for MWLP will also be given.

1.2 Problem Definition and Background

As introduced in Section 1.1, MWLP is derived from MLP. In this subsection, I briefly present the background of MLP and the algorithms proposed for it. Then, I present a brief history of MWLP. I will finish by stating the definition of MWLP.

1.2.1 Background for MLP

MLP, the base of MWLP, is usually considered as a variant of TSP. TSP is defined as the problem of finding a Hamiltonian tour that minimizes the tour length. MLP is defined as the problem of finding a path that minimizes the sum of the points' latencies, where a point's latency is its distance from the start point along the tour [6]. MLP can be formulated as follows.

Given an edge-weighted graph G_n with n vertices and a fixed start point v_1, suppose that T is a tour in G_n and use $L(v_1, v_i)$ to denote the path length from vertex v_1 to vertex v_i on tour T. The total latency is defined as

$$L(T) = \sum_{i=2}^{n} L(v_1, v_i) \tag{1.1}$$

The objective of MLP is to find a tour T_0 such that $L(T_0)$ is minimized.

There are two main differences between MLP and TSP. First, the objective of MLP is to minimize the sum of latency from the starting point, which means a fixed starting point is needed in MLP. Second, TSP tries to find a Hamiltonian tour, but for MLP, the tour is not limited to being Hamiltonian. MLP is referred to as the travelling repairman problem (TRP), delivery man problem, and school-bus driver problem in some literature. MLP is NP-complete in general graphs [6].

MLP has attracted attention in recent years, as shown in Figure 1.1.

In the same way as other NP-complete problems, three main solution approaches have been studied to solve MLP: exact algorithms, approximation algorithms, and heuristic algorithms.

For the exact algorithm, several algorithms proposed by Wu *et al.* [7] were based on branch and bound and dynamic programming (DP) strategies in 2004. Several other exact algorithms based on integer linear programming were also proposed by Luna and Sarubbi [8] and Ezzine *et al.* [9]. Ban *et al.*[10] gave a relatively efficient algorithm using a branch and bound approach in 2013. These exact algorithms are guaranteed to find the optimal solution. The shortcoming is take they exponential time in the worst case. That means they can only be guaranteed to produce results for small problem instances (up to 40 vertices) though they run quickly in practice.

The best-known approximation algorithm was given by Chaudhuri and Godfrey [6] with an approximation ratio of 3.59 in 2003. Their algorithm was based on an algorithm with an approximation ratio of 3.59α (here α is the approximation ratio of the best *k*-MST algorithm) proposed by Goemans and Kleinberg [11] by using *k*-MST algorithms in 1998. Before that, the best approximation algorithm was proposed by Blum and Chalasani [12].

Heuristic algorithms are a set of algorithms widely used in recent years. More details of heuristic algorithms are given in Chapter 3. They have been proven to work well in practice. The first heuristic approach for MLP was proposed by Salehipour *et al.*in 2008 [13], which is greedy randomized adaptive search procedure (GRASP) + variable neighbourhood descent (VND). They improved their algorithm to GRASP + VND/variable neighbourhood search (VNS) in 2011 [13]. GRASP was proposed by Feo and Resende in 1995 [14]. It can be considered as a combination of a greedy algorithm and local search. VND is a variation of VNS, and is introduced in Chapter 3. Ban and Nghia [15] applied genetic algorithm (GA; a heuristic algorithm, details are given in Chapter 3) to solve MLP in 2010. In 2013, Ban proposed a composite algorithm consisting of

GRASP, tabu search (TS; a heuristic algorithm, details are given in Chapter 3) and VNS to solve MLP [16]. Silva *et al.* [17] proposed an approach called GILS-RVND based on GRASP in 2012. Most of these heuristic algorithms are based on GRASP and VNS; only some trivial modifications have been applied.

Compared with the exact and approximation algorithms, heuristic algorithms have only been studied recently. Researchers have studied how to solve MLP by using exact and approximation algorithms for decades. However, the first heuristic algorithm for MLP was only proposed in 2008. Nevertheless, as MLP has attracted more attention in recent years, more heuristic algorithms on MLP are expected in the future.

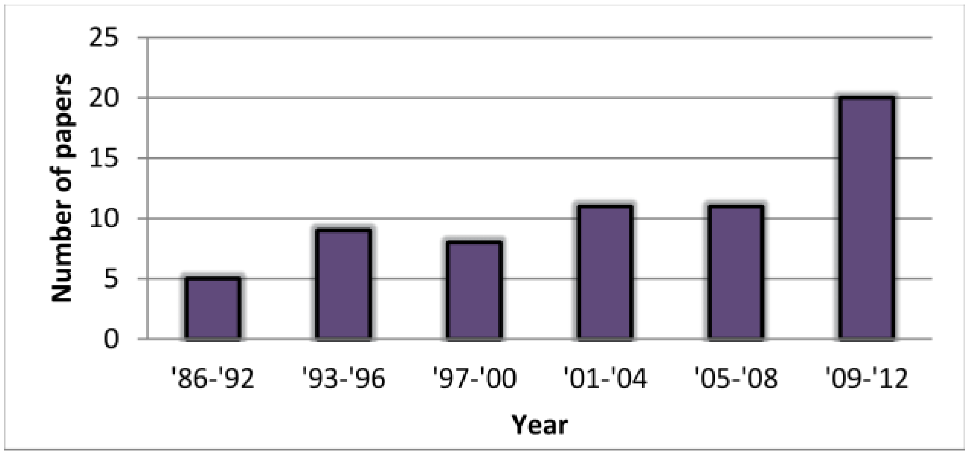

Figure 1.1: MLP research trend. From [18].

1.2.2 A Brief History and Solutions for MWLP

In 1996, Koutsoupias *et al.* [19] described the following situation. A treasure is hidden in a certain cave in a mountain area. An adventurer is trying to find the treasure. However, the mountain area is full of different caves and they all look alike. The good news is that the adventurer has a treasure map that marks the possibility of whether a given cave has the treasure and the distances between different caves. The adventurer's goal is to find a route that minimizes the distance before finding the treasure. The problem the adventurer is facing is called the graph searching problem (GSP). In [19], Koutsoupias *et al.* gave the formulation

$$\min_{\pi} \sum_{v \in G} pr(v) d_{\pi}(r, v) \tag{1.2}$$

in which π represents a tour and $d_{\pi}(r, v)$ represents the distance between the fixed starting vertex r and vertex v in tour π. Here $pr(v)$ represents the probability (weight) of a vertex v and $pr(v)$ represents the probability that the treasure is hidden at a vertex. Koutsoupias *et al.* also proved the problem is polynomial equivalent to MLP. Owing to their proof, MWLP can be easily converted into MLP in polynomial time. That means all the MLP algorithms can be used for MWLP. However, the conversion to MLP can greatly increase the size of the search domain, which dramatically reduces the efficiency of the algorithm. Koutsoupias *et al.* did not provide an algorithm originally designed for GSP. After showing the proof, they turned their focus to TRP. I explain the details of their proof in Chapter 2.

Following Koutsoupias *et al.*, Wu [20] renamed the problem MWLP in 2000 and solved it by using DP. Wu also investigated MWLP with k servers and the optimal-origin repairman problem. The problem of MWLP with k servers is simply finding k tours $(T_1, \ldots, T_k)$ that cover all the nodes in the graph. The objective is to minimize $\sum_{i=1}^{k} L(T_i)$. The optimal-origin repairman problem is used to find the origin node that minimizes the total weighted latency. Sitters [21] proved that MWLP is NP-hard in 2002. He also stated that the optimal-origin repairman problem can be solved by repeatedly running any MLP algorithm n times. Tulabandhula *et al.* [22] presented a variant of MLP, ML&TRP (machine learning and TRP). The problem is to minimize the cost of failures at vertices. Under an extreme situation, ML&TRP has the same objective function as MWLP. The authors solved the problem by splitting it. First, they used a method from artificial intelligence to predict the failure ratios of vertices. Second, the failure ratios of vertices were used to reconstruct the problem. Finally, they solved the newly generated problem by using an integer linear programming algorithm proposed in [23]. In 2008, Liu [24] proposed a DP method and several elementary heuristic algorithms for TSP-WOCT (TSP with weighted order completion times). TSP-WOCT is a problem very similar to MWLP. Further explanation of this problem is given in Chapter 3.

1.2.3 Definition of MWLP

We now come to the formulation of MWLP. First, I provide the definition of a vertex's weighted latency. It is defined as the product of the vertex's latency and its given weight. The objective of MWLP is to find a path starting at a fixed starting vertex and visiting all the vertices at least once, which minimizes the sum of the weighted latencies of all the vertices. It is formulated as follows.

Given a vertex-weighted and edge-weighted graph G_n with n vertices and a fixed starting point v_1, the vertex weights are represented by w_i for vertex v_i. Suppose that T is a tour in G_n, where $L(v_1, v_i)$ denotes the path length from vertex v_1 to vertex v_i on tour T, and $w_i L(v_1, v_i)$ is the weighted latency. The total weighted latency is defined as

$$wL(T) = \sum_{i=2}^{n} w_i L(v_1, v_i) \tag{1.3}$$

I must declare two preconditions here before I explain the experiment environment settings and go on to the main part of my thesis.

First, in this thesis, the weights are restricted to the interval $(0, 1]$ because of the study of Koutsoupias *et al.* described above. Unit normalization can be used to map the values in other ranges into this interval. This is critical because some of the algorithms that are explained later are based on this precondition. In addition, this conversion greatly simplifies the problem complexity, which makes it easier to study.

Second, in the formalized definition of MWLP, I limit the problem to a group that has a fixed starting vertex. I added this precondition because the DP algorithm needs a fixed starting vertex in the initialization phase. However, in real-world applications or theoretical research, the problem of finding a path with no fixed starting vertex is much more common. Thus, without loss of generality, I offer a method to convert the problems without a fixed starting vertex to the problems having a fixed starting vertex. The method just adds one virtual vertex to the map. The virtual vertex is set to be the starting vertex, thus its weight value is trivial. It has a distance of 0 to all other vertices on the map. By adding this virtual vertex, the original MWLP is converted into one with a fixed starting vertex, but all the other vertices' weights and latencies remain the same.

1.3 Experiment Environment Settings

Different algorithms are applied in this thesis. The DP algorithm proposed by Wu is implemented and used as the comparison group to the classic heuristic algorithms. Then the classic algorithms are used as the comparison group to the meta-heuristic algorithms proposed later. All the algorithms proposed in this thesis are implemented and tested. To maintain consistency, we run all of them in the same experimental environment. They are all coded and operated in Matlab© 2015b and tested on a small-scale server. The server is equipped with two Intel® Xeon® E5-2670 v3 @

2.30 GHz CPUs and 64 GB RAM.

The map of MWLP is obtained by randomly generating vertices in a 200×200 area. The distances between the vertices obey the triangle inequality, thus the optimal solution must be a Hamiltonian path. The weights of vertices are allocated randomly. They have 7 significant digits. Only one exception existed when we performed the experiments with 20 vertices. We tested two groups of data. The weights of vertices of one of the groups is ordinarily randomly allocated. The weights of vertices of the other group are randomly selected from a weight number pool containing five different numbers. They are 0.2, 0.4, 0.6, 0.8, and 1 respectively. This somewhat simplified the problem. I designed this experiment to test the computational limitations of the heuristic algorithms. Details are given in Chapter 3.

1.4 Organization of the Thesis

This thesis contains five chapters in total. In Chapter 2, I introduce the existing exact algorithms and approximation algorithms. An integer programming algorithm is also given. In Chapter 3, I describe some existing heuristics for a problem called TSP-WOCT, which is very similar to MWLP. Several chosen classic heuristics will then be implemented to solve MWLP. The test results and the theoretical analysis are also given. In Chapter 4, I introduce two meta-heuristics containing a heuristic strategy called VNS and a heuristic strategy called P system. The two strategies are introduced, and the methods based on the ideas of VNS and P system are also explained. Tests are performed to illustrate the searching capacity of these two meta-heuristics. I introduce the process of simulating the method and analyse the effectiveness and efficiency of the algorithms. In Chapter 5, I briefly introduce two real-world applications of the mathematical model. They are the perishable goods transportation problem and a model of the disaster relief problem. Some future work is also discussed.

2 Exact Algorithm for MWLP

As mentioned in the previous chapter, both exact and approximation algorithms have been applied to MLP. Heuristic methods have not been applied as widely. Koutsoupias *et al.* proved that as long as the weights of the vertices of the map are rational numbers, MWLP can be converted into MLP [19]. Thus, any of the existing algorithms for MLP can be used to solve the original MWLP instance by solving the newly generated MLP instance. Theorem 2.1 is the basis for the reduction, as proposed by Koutsoupias *et al.*

Theorem 2.1 (Koutsoupias *et al.*'s Theorem 3 [19])**.** *If the distances in the metric d are polynomially small integers, and the probabilities* ***pr*** *are rational numbers with small coefficients and common denominators, the two problems are polynomially equivalent.[19]*

The authors proved the theorem by simulating a vertex with weight A/B by a cluster of A vertices with zero distance from each other, where B is the common denominator. Because B is the common denominator, all the weights of the vertices can be multiplied by B to become integers. After this manipulation, the weighted latencies of the vertices are multiplied by B, as well as the final result. This result can be easily used to calculate the real value (the sum of the weighted latencies of the vertices before the manipulation). Denote the latency of the vertex as L, its weighted latency is AL. Now simulate the vertex with a cluster of A vertices with weights one and zero distance from each other. Because the distances between the vertices are zero, the latencies of all these A vertices are L. The sum of the weighted latencies of these A vertices is AL, which is equal to the original one.

Figure 2.1 shows an example of this proof. Suppose we are given $w(A) = w(B) = w(D) = 1/5$ and $w(C) = 3/5$. For tour $T = (A, B, C, D)$, $L(T, A) = 0$, $L(T, B) = 3$, $L(T, C) = 8$, and $L(T, D) =$

14. Thus, $wL(T) = L(T,A)\times 1/5 + L(T,B)\times 1/5 + L(T,C)\times 3/5 + L(T,D)\times 1/5 = 41/5$. During the conversion process, all the vertices' weights are multiplied by five at the beginning. They become $w(A) = w(B) = w(D) = 1$ and $w(C) = 3$. According to the new weights, nodes A, B, D become nodes A', B', D', and their weights are $w(A') = w(B') = w(D') = 1$. Node C becomes the three nodes $C1'$, $C2'$, and $C3'$. Their weights are one and the distances between them are zero. For the new tour $T' = (A', B', C1', C2', C3', D')$, $L(T, A') = 0$, $L(T, B') = 3$, $L(T, C1') = 8$, $L(T, C2') = 8$, $L(T, C3') = 8$, and $L(T, D') = 14$. Thus, $wL(T') = L(T, A') \times 1 + L(T, B') \times 1 + L(T, C1') \times 1 + L(T, C2') \times 1 + L(T, C3') \times 1 + L(T, D') \times 1 = 41$. Here $wL(T') = wL(T) \times 5$.

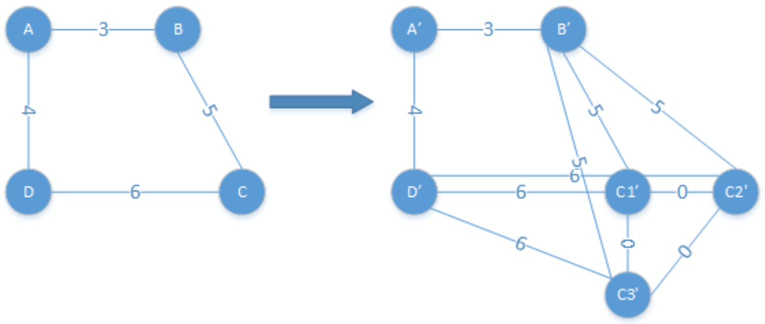

Figure 2.1: Illustration of Theorem 2.1.

This idea is brilliant. It reduces the difficulty level of the problem by eliminating the weights of the vertices, one of the two variables in MWLP. Nevertheless, the conversion causes some new shortcomings.

By using this method, the problem must meet a precondition that the weights are rational numbers. Otherwise, it is impossible to compute the new weights. Second, the weights must be numbers with small coefficients and common denominators. This is a requirement proposed from the theoretical aspect. However, in practice, these limitations are hard to meet. The latter precondition requires the weights of the vertices to be numbers with no large significant figures. This number determines the degree to which the problem scale expands after the conversion. With only one increase of the significant figure number, the problem will grow by a factor of 10 compared with the original version. In some practical applications, the data precision can be very high. This dramatically increases the problem's scale after conversion.

In this chapter, I introduce two existing dynamic programming (DP) algorithms. They are designed for MWLP and TSP-WOCT, respectively. TSP-WOCT is a variety of MWLP. In some extreme condition, it is equal to MWLP. After the introduction, I describe a mixed integer programming algorithm for MWLP.

2.1 Existing Algorithms

To the best of the author's knowledge, only two exact algorithms for MWLP have been proposed, the first by Wu [20] and the other by Liu [24]. The latter was initially designed as an algorithm for TSP-WOCT, which can be seen as a special case of MWLP. Thus, without much revision, the algorithm can fit MWLP quite well.

Both methods are derived from DP, but they used different methods to divide the given problem into subproblems. Wu did it by reconstructing the graph. The graph is cut into small parts (subgraphs). The subgraphs are connected by "cutting nodes". After the visiting sequence has been determined in one subgraph, the subgraph is considered as a vertex in the whole graph. Liu reconstructed the objective function. The algorithm traverses all the visiting sequences to find the best solution. Liu redesigned the format of the objective function to help him compute the value of the "next" sequence with time complexity O(1).

2.1.1 Wu's Algorithm

As described above, Wu's paper was the first to define MWLP, though some of the ideas are sketchy from today's perspective. He was trying to find a tour starting at a fixed starting point and visiting all the vertices to minimize the sum of the weighted latencies of the vertices. In addition, he discussed the problem of finding multiple ways (multiple mobile agents) in one graph and the problem of looking for the best start point that can minimize the sum of weighted latencies. To use DP to solve MWLP, he first defined a cost function, $c(G, P)$, as follows.

Definition 2.1 (Wu's Definition 3 [20])**.** *Let P be a subtour on graph G. Define $c(G, P) = L(P) + (w(G) - w(P))d(P)$.*

Here, $L(P)$ is the sum of weighted latencies of all the vertices on subtour P, $w(G)$ is the sum of all the weights of the vertices in graph G, $w(P)$ is the sum of all weights of the vertices on subtour P, and $d(P)$ denotes the length of subtour P. When P passes all the vertices in graph G, $c(G, P)$ is exactly the sum of all vertices' weighted latencies of P on G, which is the objective of the problem.

Then he proposed a way to divide a graph into small graphs (subgraph) by using a "cut node".

Definition 2.2 (Wu's Definition 6 [20])**.** *Let v be a cut node of a connected graph G and G_1, G_2 be connected subgraphs of G. Then G is split into G_1 and G_2 at v if $V(G_1) \cap V(G_2) = \{v\}$ and the union of G_1 and G_2 is G.*

Wu used a tuple (l, r) to represent a subtour visiting m points. In the tuple, the first unit l is a k-component integer vector, $l = (l_1, l_2, \ldots, l_k)$. In the vector, k represents the number of subgraphs and l_i represents the number of nodes (l, r) visits in subgraph i. The second unit of the tuple is an integer $r \leq k$. It indicates the subgraph where the subtour stops. Here $T(l, r)$ is defined as the minimum $c(G, P)$ of all the subtours represented by tuple (l, r). If P visits all the vertices in G, $T(l, r)$ is equal to the minimum weighted latencies.

By using this representation scheme, the objective function can be reconstructed in a different way. Now increase l's rth component by one node to obtain a new vector l'. That means tuple (l', r) ends at a vertex in subgraph r. The vertex has to satisfy two requirements: the first is that it has not been visited by tuple (l, r), and the second is that it is adjacent to tuple (l, r)'s last vertex. The value of $T(l', r)$ can be computed by the following DP formula:

$$T(l', r) = \min_{1 \le i \le k} \{T(l, i) + (w(G) - w(V(l)))d_G(v_i, u)\} \tag{2.1}$$

where $V(l)$ is the set of vertices visited by subtours represented by l. By using this formulation, a DP algorithm can be obtained in a very straightforward manner. If the order of the nodes visited within a subgraph is known, MWLP can be solved.

Then, Wu presented a method to determine the visiting order when the subgraphs are trees or stars. When it comes to a tree with k leaves, he split it into k paths and nodes with degrees larger than two at the origin. After performing this manipulation, the visiting order of the nodes is fixed. Regarding stars, a star can be cut into several subgraphs at the internal node. Because the subgraphs are paths, the visiting orders within the subgraphs are also fixed. The visiting sequence of the subgraphs can be determined by using the following lemma.

Lemma 2.1 (Wu's Lemma 2 [20]). *Assume H is a star split from graph G at the internal node p of H. For any leaves i and j of H, i will be visited before j on the optimal tour if $w(i)/d(p, i) > w(j)/d(p, j)$, and the tie can be broken arbitrarily.*

By computing the weight and distance ratio, the visiting order can be calculated.

2.1.2 Liu's Algorithm

TSP-WOCT originates from a satellite imaging problem. Assume a satellite is dealing with several requests, where all these requests are asking the satellite to take photos of several different places on Earth. For the satellite, different requests have different urgency (priority). For example, a request from the Ministry of Defence can be more important than a request from a commercial corporation whose main business is a geographic information system (GIS). Because the locations of the requested photographs are different, the satellite has to adjust the lens angle before taking the shot. This operation needs to be extremely accurate, which means it takes much longer than simply clicking the shutter. The satellite will take all the photos to complete the requests sequentially. The completion time of a request is the time when all the photos of this request have been taken. The scheduling objective is to minimize the sum of the requests' completion times multiplied by their urgencies.

The TSP-WOCT problem can be formulated as follows. A single machine is going to handle K orders $\sigma = \{O_1, O_2, \ldots, O_K\}$. Order O_k consists of n_k jobs and is allocated a non-negative weight w_k. Define $n = n_1 + n_2 + \cdots + n_k$. Here N is the set consisting all the jobs from all the orders, $N = \bigcup_{k=1}^{K} O_k = \{J_1, J_2, \ldots, J_n\}$. The setup time of job J_j is d_{ij} if it follows job J_i. A job's processing time is negligible and the machine's starting status is the initial job J_0. The completion time of order O_k is C_k, which is the time when O_k's last job is done. The objective is to find a processing sequence that minimizes the sum of all the orders' weighted completion times $\sum_{k=1}^{K} w_k O_k$.

If all the jobs turn into orders, namely every order contains only one job, TSP-WOCT turns into MWLP. That is, MWLP is a special instance of TSP-WOCT. Thus, the method used to solve TSP-WOCT can also be used to solve MWLP. Liu proposed a DP algorithm that is believed to be inherited from Wu's algorithm.

Liu gave his formulation as [24]

$$W(S(N', J_j)) = \sum_{O_k \in \theta(N')} w_k C_k(S(N', J_j)) + L(S(N', J_j)) \times \sum_{O_k \in \theta \backslash \theta(N')} w_k \tag{2.2}$$

where N' denotes a subset of jobs, $S(N', J_j)$ denotes all the sequences in N', J_j is the last scheduled job, $W(S(N', J_j))$ is the sum of weighted latencies, w_k and $C_k(S(N', J_j))$ are the weight of one order in $S(N', J_j)$ and its cost (length), $\theta(N')$ denotes the orders already completed in N', and $L(S(N', J_j))$ denotes the length of sequence $S(N', J_j)$.

When all the orders are completed, the second term in the first sum equals zero. At that point, $W(S(N', J_j)) = \sum_{O_k \in \theta(N')} w_k C_k(S(N', J_j))$.

Liu provided the example listed in Table 2.1.

$S_1 = J_2J_4J_3J_1$	$S'_1 = J_2J_4J_3J_1J_5$	$W(S'_1) = W(S_1) + d_{15} \times \sum_{O_k \in \theta \backslash \theta(N')} w_k$
$S_2 = J_4J_3J_2J_1$	$S'_2 = J_4J_3J_2J_1J_5$	$W(S'_2) = W(S_2) + d_{15} \times \sum_{O_k \in \theta \backslash \theta(N')} w_k$

Table 2.1: Example from Liu [24].

Table 2.1 can be obtained quite straightforwardly and leads to two conclusions.

1. Assume that the latency of a sequence S_1 is smaller than a sequence S_2, and both select the same job to be next. The weighted latency of the sequence generated by S_1 is still smaller than that of the sequence generated by S_2.

2. The weighted latency of the new sequence can be easily computed by using the old one's value.

By using the two conclusions above, Liu provided [24]

$$f(N', J_j) = \min_{S(N', J_j) \in \prod_{(N', J_j)}} \{W(S(N', J_j))\} \tag{2.3}$$

where $\prod_{(N', J_j)}$ denotes the set of all $S(N', J_j)$. By computing the value of this function, TSP-WOCT can be solved by using DP.

2.1.3 Comparison

Figure 2.2 shows a map with six vertices, which can be used to illustrate the difference between MWLP and TSP-WOCT.

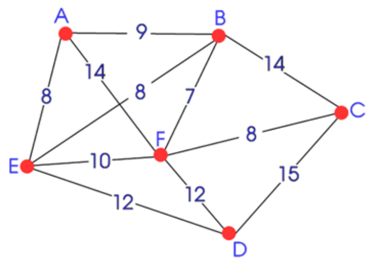

Figure 2.2: Demonstration of MWLP and TSP-WOCT, from [25].

In terms of MWLP, each vertex is allocated a weight: $w(A) = 0.2$, $w(B) = 0.3$, $w(C) = 0.5$, $w(D) = 0.9$, $w(E) = 0.4$, and $w(F) = 0.2$. For sequence $S = (A, B, C, D, E, F)$, Table 2.2 lists the latency value and weighted latency value of each vertex.

	A	B	C	D	E	F
Latency	0	9	23	38	50	60
Weighted Latency	0	2.7	11.5	34.2	20	12

Table 2.2: Latency and weighted latency for $S = (A, B, C, D, E, F)$.

As for TSP-WOCT, the six vertices are divided into two orders $O_1 = (A, C, F)$ and $O_2 = (B, D, E)$. The weights for O_1 and O_2 are 0.3 and 0.4, respectively, and we consider the same sequence $S = (A, B, C, D, E, F)$. The latency of O_1 is the latency of the last visited vertex in O_1, which is F, thus 60. The weighted latency of O_1 is $60 \times 0.3 = 18$. Similarly, the latency of O_2 is the latency of E, which is 50. The weighted latency of O_2 is $50 \times 0.4 = 20$.

Both Wu and Liu used the idea of DP. As mentioned previously, Liu's algorithm can be seen as derived from Wu's.

The two algorithms have some aspects in common as follows.

1. If we take the jobs as graph nodes and the preparation time as the distance between different nodes, TSP-WOCT can be treated as a problem of minimizing the total weighted latencies of the sets of nodes.

2. The forms of objective functions and patterns of organizations are almost the same. Liu designed his formula by borrowing ideas from Wu. The only difference is that Liu changed the sum of the weights of the nodes into the sum of the weights of the orders.

Nevertheless, some differences exist. They can be seen as evolutions of Liu's algorithm.

1. The two algorithms are dealing with two different problems. TSP-WOCT tries to minimize the total weighted latencies of the orders, not the jobs, whereas MWLP is minimizing the total weighted latencies of the nodes.

2. In TSP-WOCT, one order consists of several jobs. In a way, this simplifies the complexity of the problem. However, Wu treated MWLP in a smart way: he focused on dealing with the subgraph and used different ways to deal with the subgraphs with different types (such as a path or caterpillar). In this way, Wu reduced the scale of the problem.

I must clarify here, at the end of this subsection, that both Wu and Liu's DP algorithms can be used as a comparison group in this thesis, as I will use a complete graph to test the proposed algorithms capability. Wu classified the subgraphs into different types to deal with them separately. However, if the graph is a complete graph, the subgraphs are separate vertices. The solution is straightforward. In terms of Liu's method, as long as every vertex in MWLP is treated as a job and also an order, the problem can be transferred perfectly as a TSP-WOCT. Then Liu's method can be used to solve MWLP directly. A more interesting consequence is that when we use this method to deal with the two algorithms, they tend to end up the same. This proves the speculation mentioned above that Liu's algorithm is the descendant of Wu's DP algorithm.

2.2 Mixed Integer Programming

Integer programming is a widely used methodology for mathematical optimization and feasibility problems. Different from other mathematical programming methods, its variables are limited to be integers during the programming process.

0–1 programming or binary integer programming (BIP) is a widely used special case among all the branches of integer programming. It requires the programming variables to be 0 or 1. Because the structure of BIP is very similar to many real-world applications, such as assignment problems, location problems, and logistics, it plays an important role in the area of mathematical programming research. In addition, any integer programming problems with bounded variables are equal to BIP following proper conversion. This helps to extend the application range of BIP. In addition, many kinds of non-linear programming problems can also be converted into BIP problems. This attracts many researchers from other research communities.

Mixed integer programming is a set of problems in which some of the variables are limited to be integers but the others are not. In this section, we present a mixed integer programming solution for MWLP.

2.2.1 Mixed Integer Programming for MWLP

As for MWLP, its mathematical model is very similar to that of TSP, which means the existing modelling methods of TSP can also be used to model MWLP. Nonetheless, some differences still exist. Modifications must be made to the existing models to fit MWLP. The main difficulties are as follows.

1. TSP is a problem that looks for a closed tour (cycle) with a minimum length. However, MWLP looks for a path. Simply speaking, MWLP eliminates an edge between the last vertex and the starting vertex of the TSP cycle. The elimination of this edge can provide a difference between the modelling methods of TSP and MWLP.

2. When dealing with TSP, not only is the graph undirected, but so is the route. That is, no matter which direction you choose from the starting vertex, the length it takes to finish a Hamiltonian cycle remains the same. In contrast, though we use an undirected graph to study MWLP, the final results will change if the direction is changed. MWLP's features require its mathematical model to be directed. This will lead to a major difference between the two models.

Nevertheless, we designed a model to handle MWLP. It is based on a model described in [26], which was used to solve TSP. The details are as follows.

In Chapter 1, we gave the formal definition of MWLP. MWLP looks for a Hamiltonian cycle starting from a fixed starting vertex. In this algorithm, we denote the vertex as v_1. Because it is the first visited vertex, its weight value is trivial (the value will be multiplied by zero anyhow). Without loss of generality, we assume there are $n - 1$ vertices other than v_1. Thus, there will be a total of n vertices in this graph. We use p to denote a path.

The following is a formal description of MWLP in natural language. It contains both the objective function and limitations of the variables. We will use it as a starting point to deduce the details of the mixed integer programming method:

$$\begin{aligned} &\min \text{ The sum of all the weighted latencies of the vertices} \\ &\text{s.t. a Hamiltonian path in graph } G \end{aligned}$$

To represent the model's objective function in mathematical language, the model becomes

$$\min \sum_{i=1}^{n} w_{(i)} Latency_{(i)} \tag{2.4}$$

$$\text{s.t. a Hamiltonian path in graph } G \tag{2.5}$$

where $w_{(i)}$ denotes the weight of the ith visited vertex of the path and $Latency_{(i)}$ denotes the latency of the vertex. To make the process of mathematical derivation clearer, we focus on the objective function from now on. The process of the variable limitations is given later.

First, we break up the $\sum$ symbol. Now the objective function is written as

$$\min w_{(1)} Latency_{(1)} + w_{(2)} Latency_{(2)} + \cdots + w_{(n)} Latency_{(n)}$$

where we use $d(i, i+1)$ to denote the distance between the ith visited vertex and the $(i+1)$th visited vertex, and use the variable W_{sum} to denote the sum of weights of all the vertices. Namely, $W_{sum} = \sum_1^n w_{(i)}$. Because $Latency_{(1)} = 0$, the objective function becomes

$$\sum_{i=1}^{n} w_{(i)} Latency_{(i)} = w_{(2)} Latency_{(2)} + w_{(3)} Latency_{(3)} + \cdots + w_{(n)} Latency_{(n)} \tag{2.6}$$

$$= w_{(2)} d(1,2) + w_{(3)}(d(1,2) + d(2,3)) + w_{(4)}(d(1,2) + d(2,3) + d(3,4)) \tag{2.7}$$

$$+ \cdots + w_{(n)}(d(1,2) + d(2,3) + \cdots + d(n-1,n)) \tag{2.8}$$

$$= d(1,2)(w_{(2)} + w_{(3)} + \cdots + w_{(n)}) + d(2,3)(w_{(3)} + w_{(4)} + \cdots + w_{(n)}) \tag{2.9}$$

$$+ \cdots + d(n-2,n-1)(w_{(n-1)} + w_{(n)}) + d(n-1,n) w_{(n)} \tag{2.10}$$

$$= d(1,2)(W_{sum} - w_{(1)}) + d(2,3)(W_{sum} - w_{(1)} - w(2)) \tag{2.11}$$

$$+ \cdots + d(n-1,n)(W_{sum} - w_{(1)} - w_{(2)} - \cdots - w_{(n-2)} - w_{(n-1)}) \tag{2.12}$$

Now, we introduce another variable $W'_{(i)}$ to simplify this equation: $W'_{(i)}$ denotes the value that $d(i-1, i)$ is multiplied by in Equations (2.11) and (2.12) or $W'_{(i)} = W_{sum} - \sum_1^{i-1} w_{(i)}$. Namely,

$$\begin{aligned}
W'_{(1)} &= W_{sum} \\
W'_{(2)} &= W_{sum} - w_{(1)} \\
W'_{(3)} &= W_{sum} - w_{(1)} - w_{(2)} \\
&\cdots\cdots \\
W'_{(n)} &= W_{sum} - w_{(1)} - w_{(2)} - \cdots - w_{(n-2)} - w_{(n-1)}
\end{aligned}$$

Obviously, $W'_{(i+1)} = W'_{(i)} - w_{(i)}$.

Now, the objective function becomes

$$\sum_{i=1}^{n} w_{(i)} Latency_{(i)} = d(1,2)W'_{(2)} + d(2,3)W'_{(3)} + \cdots + d(n-1,n)W'_{(n)} \tag{2.13}$$

$$= \sum_{i=2}^{n} d(i-1,i)W'_{(i)} \tag{2.14}$$

Now, we have almost finished the reconstruction of MWLP's objective function. However, it is still not sufficient to represent MWLP in an appropriate mixed integer programming model. In the following, we introduce a mathematical tool called the value system. This can help us to further deal with both the objective function and the restrictions. The value system is mainly an $n \times n$ matrix X consisting of zero and one. Its element X_{ij} $(1 \leq i \leq n, 1 \leq j \leq n)$ represents whether there exists an arc from vertex i to vertex j in path p. If the arc exists, the element X_{ij} is set to one. Otherwise, X_{ij} is zero. MWLP does not consider the situation where a vertex sends out an arc to itself. Thus, as for every vertex i, $X_{ii} = 0$. The value system is introduced to help represent the paths on a map. Matrix X can represent both the objective function and any path that satisfies the requirement of MWLP in mathematical language.

As for the objective function, the major problem is how to represent $d(i-1,i)$ and $W'_{(i)}$ by using the value system. Noting that the objective function result depends on every vertex except the first visited one, which means every vertex computed in the objective function has a pre-visited vertex. Considering a vertex v_i other than v_1. By using matrix X, the length of the arc visited before v_i, $d(i-1,i)$, can be represented by

$$d(i-1,i) = \sum_{j=1}^{n} d(j,i)X_{ji}$$

This is because as for any vertex v_j in graph G other than v_i, if there exists an arc from it to v_i, $X_{ji} = 1$, and $d(i-1, i) = d(j, i) = d(j, i) \times X_{ji}$. If v_j is not visited immediately before v_i in path p, $X_{ji} = 0$. This makes the product of $d(j, i)$ and X_{ji} equal to zero. Because $X_{ii} = d(i, i) = 0$, the sum of all the products of $d(j, i) \times X_{ji}$ for a single vertex v_i equals $d(i-1, i)$.

Following the same idea, and combining this with the fact that $W'_{(i)} = W'_{(i-1)} - w_{(i-1)}$, the value $W'_{(i)}$ can be represented as

$$W'_{(i)} = \sum_{j=1}^{n} (W'_{(j)} - w_i) X_{ji} \tag{2.15}$$

Now the objective function can be converted into

$$\sum_{i=1}^{n} w_{(i)} Latency_{(i)} = \sum_{i=2}^{n} \Big(\sum_{j=1}^{n} d(j, i) X_{ji} * W'_{(i)} \Big)$$

The objective function has now been converted successfully. In the next step we to show the method to present the restrictions of the variables. In other words, we demonstrate how to use the value system to represent a Hamiltonian path.

First, we have a fixed starting vertex v_1. Its in-degree is zero and out-degree is one:

$$\sum_{j=1}^{n} X_{jv_1} = 0$$
$$\sum_{j=1}^{n} X_{v_1 j} = 1$$

Second, as for any other vertex i that is not the starting vertex, its in-degree is one and its out-degree is at most one:

$$\sum_{j=1}^{n} X_{ji} = 1$$
$$\sum_{j=1}^{n} X_{ij} \leq 1$$

Third, some other limitations are still required to eliminate the possibility of the existence of a simple cycle. Here we introduce a proposition proposed by Feng [26]. We did not find an English version proof of the proposition, so we also present the method of proof here.

Lemma 2.2. *There is no simple circle in path p if and only if there exists*

$$u_i \in \{0, 1, 2, \ldots, n-1\}, \quad i = 1, 2, \ldots, n$$

which makes

$$u_i - u_j + nX_{ij} \leq n - 1, \quad i \neq j, \ (i, j) \in p$$

We now prove Lemma 2.2. First, we prove its necessity.

Proof. 1. If $X_{ij} = 0$, the inequality becomes $u_i - u_j \leq n - 1$. For every $i = 1, 2, \ldots, n$, $u_i \in \{0, 1, 2, \ldots, n-1\}$. The maximum in u_i is $n - 1$ and the minimum is zero. The inequality is proved.

2. If $X_{ij} = 1$, the inequality becomes $u_i - u_j \leq -1$. Since $X_{ij} = 1$, the arc between vertex i and vertex j is chosen in p, and vertex i is visited before vertex j. We only have to find a permutation of $u_i \in \{0, 1, 2, \ldots, n-1\}$ to satisfy the inequality to prove this part. The

vertices other than v_1 have in-degrees of one and out-degrees of zero or one. Combined with the precondition that there is no simple circle, we know path p must be a Hamiltonian path. Here, we set the root v_1's u value $u_{v_1} = 0$. The second visited vertex's $u = 1$, and so forth. In other words, we set the vertices' u values to be their visited orders minus one. Since vertex i is visited exactly one vertex before vertex j, we obtain $u_i - u_j = -1$. The inequality is proved.

□

Now we prove the sufficiency.

Proof. We prove sufficiency by contradiction.

Assume there exists a set of u_i which satisfies the inequality above, but p still has at least one simple circle. We assume the circle to be $(v_{p1}, v_{p2}, \ldots, v_{pm}, v_{p1})$. Since it is a circle, $m \geq 2$. Within the circle, all the vertices are connected one by one. Hence, we have

$$X_{p1p2} = X_{p2p3} = \cdots = X_{pmp1} = 1.$$

By putting this into the inequality, we have

$$\begin{aligned} u_{p1} - u_{p2} &\leq -1 \\ u_{p2} - u_{p3} &\leq -1 \\ &\vdots \\ u_{pm} - u_{p1} &\leq -1 \end{aligned}$$

By adding all these inequalities together, we have $0 \leq -m$. Since $m \geq 2$, the inequality is not sufficient. Hence, the hypothesis is not satisfied. The sufficiency is thus proved. □

Finally, by adding the last restriction, we can complete this mixed integer programming. It is the definition of $W'_{(i)}$, which includes the value of $W'_{(1)}$ and Equation (2.15). It is also the second factor other than the value system in this mixed integer program:

$$\begin{aligned} W'_{(1)} &= W_{sum} \\ W'_{(i)} &= \sum_{j=1}^{n} (W'_{(j)} - W_i) X_{ji}, \quad i \neq v_1 \end{aligned}$$

By combining the restrictions and objective given above, the normalized form of the mixed integer program is given as follows:

$$\min \sum_{i=2}^{n} \Big(\sum_{j=1}^{n} d(j,i) X_{ji} W'_{(i)} \Big)$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{n} X_{ji} = 0, & i = v_1 \\ \sum_{j=1}^{n} X_{ji} = 1, & i \neq v_1 \\ \sum_{j=1}^{n} X_{ij} \leq 1, & i = 0, 1, \ldots, n-1 \\ u_i - u_j + nX_{ij} \leq n - 1, & u_i \in \{0, 1, 2, \ldots, n-1\} \\ X_{ij} = 0 or 1, & \forall i \text{ and } \forall j \in G \\ W'_{(1)} = W_{sum} & \\ W'_{(i)} = \sum_{j=1}^{n} (W'_{(j)} - W_i) X_{ji}, & i \neq v_1 \end{cases}$$

3 Classic Heuristic Algorithms for MWLP

In this chapter, I focus on the feasibility of using classic heuristics to solve MWLP. Five classic heuristic algorithms are first briefly introduced and then I explain how to use these heuristics to solve MWLP.

Heuristic algorithms have drawn attention from different research communities and have been used to solve many problems in recent years. Researchers have proved that heuristics can be implemented for many NP-hard problems, where they can obtain good results in an acceptable runtime. Among the heuristics, genetic algorithms (GA), simulated annealing (SA), particle swarm optimization (PSO), tabu search (TS), and ant colony optimization (ACO) algorithms are most widely used[27]. Thus, I chose them to test the capability of heuristics when solving MWLP. These five algorithms have different features that can represent both the advantages of heuristics but also their limits to some extent. The heuristics for MWLP are designed and implemented based on their basic ideas with only minor modifications and specifically tuned parameters for different problem cases. More details will be given in the following.

The MWLP instances were divided into three types by the number of vertices of their maps. The problem with graphs having fewer than 30 vertices was considered a small-scale problem. Similarly, the problems with 30–50 vertices were considered as medium-scale problems. Problems with more than 50 vertices were considered as large-scale problems.

The parameter settings were designed based on the small- and medium-scale problems. Nevertheless, the experimental results showed they also fit the large-scale problems. Compared with exact algorithms, the experimental results showed that the tested classic heuristics perform more efficiently and effectively.

3.1 Heuristic Algorithms

As mentioned above, five kinds of heuristics were applied to solve MWLP. Here, I give a brief introduction to the development of heuristics and then the explanations of these heuristics and the basic parameter configuration are introduced.

3.2 Development of Heuristics

Heuristic algorithms are a type of algorithm generated from natural operation laws (like GA) and real working experiences (like TS). When dealing with an NP problem, the actual usability of exact and approximation algorithms can be very limited. In practice, a "reasonable" result is good enough. That is where heuristics can help.

Heuristics are designed to solve NP problems, and the heuristic mechanism does a good job of reducing the amount of computation. However, the amount of computation is still huge. That is, heuristics also require powerful computing platforms. This explains why the progress of heuristic algorithms is accompanied by the development of computer hardware.

In the 1940s, heuristic algorithms were proposed for the first time because they are fast and applicable. In the next decade, heuristics became numerous: local search and greedy algorithms drew attention from researchers. In the 1960s, people found that heuristics have many shortcomings. For example, the quality of the results is hard to guarantee. What is worse, the search speed is too slow when dealing with large-scale problems. Compared with exact and approximation algorithms, heuristics were seen to have no remarkable advantages. This was a dark time in the history of heuristic development. During the 1970s, great developments in the theory of computational complexity had been made. Many problems had been proved to be "hard". That means it is hard to find the optimal solution of these problems in polynomial time. On the other hand, the heuristic algorithm was developing quickly. Researchers had figured out that by expanding the search domain, the heuristics can jump out of the local optimum. After the 1980s, new heuristic algorithms were developed, including SA, TS, artificial neural network (ANN)[28, 29], and so on. Recently, new heuristics such as the evolutionary algorithm (EA) [30, 31], ACO, P system (membrane computing), and quantum computation (QC) [32, 33, 34] have been designed.

3.2.1 Genetic Algorithm

Professor J. Holland proposed GA in 1975. He published his book *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology* in 1992[35]. In this book, he systematically described the idea of GA. The algorithm is inspired by the biological evolution process. It simulates the choice during the combination of different types of DNA, along with the cross-breeding process and the mutation of chromosomes. During the evolution process, the individual with higher fitness has a higher probability of surviving. In recent years, with the development of biotechnology and the demand for efficient and effective optimization methods for large-scale problems, GA has drawn increased attention [36, 37, 38].

GA simulates the process of natural evolution in the light of the evolution theory at the chromosome level. GA produces better solutions to the optimization problem based on the principle of survival of the fittest. A string of binary values referred to as the chromosome is used to encode each individual. In the process of GA, three randomized operations are applied. They are selection, cross-over, and mutation. At the start of the algorithm, a series of candidate solutions are generated randomly. Next, “selection” is applied. It is a process of choosing the chromosomes from the previous generation populations, which are used to produce offspring for the next generation. Many different techniques are suitable for this operation, such as rank selection, tournament selection, and roulette wheel selection. The operation "cross-over" takes two random parent chromosomes from the population formed by "selection" to generate two offspring chromosomes. The operation "mutation" is executed after cross-over. Parts of the parent chromosomes are randomly changed according to a prefixed mutation ratio in this operation. This prevents the algorithm being trapped in local optima. After repeatedly performing these three operations, the fitness of each individual in the new populations is improved. The optimization process continues until some stopping criterion is satisfied.

Usually, three stopping criteria are set for GA. The optimization process stops if any of them is reached or passed. They are:

1. value of the best-fitted individual's fitness;
2. number of the generations the best individual or the whole population's mean fitness value remains unchanged;
3. number of the evolution generations.

The second criterion is chosen based on the goal that the optimization process is supposed to reach.

Through testing, the parameters of GA were chosen as follows. The population number was set to 100. The selection ratio and cross-over ratio were both set as 0.9, and the mutation probability was 0.05.

Regarding the stopping criteria, we dismissed the first and the second parameters. Under normal circumstances, we could not know a problem's global optimum before we run any heuristic.

In addition, in theoretical research, it is usually impossible to know the range of a problem's global optimum before we test it. This is the reason why we dismissed the first stopping criterion parameter. In our experiments, the numbers of iterations were set before the test. We would like to find out how much time the algorithms consume when their numbers of iterations vary. Thus, the second parameter was also dismissed. The generation number was set to 50 after the experiments.

3.2.2 Simulated Annealing

According to [39], Metropolis *et al.* initially proposed the basic idea of SA in 1953. It was introduced into the research community of optimization by Kirkpatrick *et al.* in 1983 [40]. It is a random search strategy based on the Monte Carlo method. Both the inspiration and name of SA come from annealing in metallurgy. The algorithm simulates the process a material experiences when cooling down from a high temperature to room temperature. During the annealing process, the material will turn from liquid to solid, and its inner structure will become more stable. At the end of the simulation, if the algorithm finds a better result, the algorithm will accept it. Otherwise, the algorithm will accept it with a certain probability. This mechanism helps the algorithm jump out of a local optimum. Recently, SA has become increasingly popular because of its advantages : it is easy to implement, has a small-scale application program, and short running time [41, 42]. In [43], Kieu even designed a SA for TSP which consumes almost fixed amount of time regardless the problem scale based on the features of quantum computation.

As mentioned above, SA is derived from the annealing process of metal. Given an initial temperature, the algorithm will produce a solution in polynomial time by slowly dropping the temperature parameter. This manipulation mimics the process of annealing a metal. When annealing metal, the molecules in a metal are moving in a disordered state. As the temperature decreases, the inner structure tends to become more well-ordered and stable. This process corresponds to finding the optimum in a search space. In our implementation, at the beginning of the search process, a random permutation of the vertices is chosen as the initial solution. The manipulation 2-OPT (short for 2-optimization: this generates a new solution by exchanging the positions of two randomly selected elements) is used to find a new solution. The new solution is called the previous solution's neighbour. The initial solution's neighbours and its neighbours' neighbours are selected as the search proceeds. The probability of choosing a solution or its neighbour is determined by using the Metropolis criterion. A commonly used Metropolis criterion is: if $\Delta T < 0$, then the new solution is accepted; otherwise, the new solution is accepted with a probability of "$\exp(-\Delta T/T)$". Here ΔT represents the metal's temperature change between an unit time. T represents the metal's temperature. The search process stops when the search space is stable. Usually, this means the system temperature has dropped to a pre-determined threshold, or the number of newly generated solutions that have not been accepted has passed a threshold.

We ran a series of experiments to select the parameters for SA. The initial temperature was set to 1000°C. The temperature was multiplied by 0.9 each time we determined whether the new solution would be chosen (we called this an iteration). The final temperature was set to 0.001°C. Similarly to what we did in GA, we also dismissed the second stopping criterion in this algorithm because the algorithm ran quickly. The increase in time and space consumption caused by dismissing one criterion is acceptable, and we would like to trade this consumption for the possibility of finding a better solution.

3.2.3 Particle Swarm Optimization

PSO was originally proposed by Eberhart and Kennedy in 1995 [44]. It is considered as one type of swarm intelligence (SI). PSO simulates a flock of birds searching for food in a certain area. The birds are considered as “particles” in the algorithm. None of the birds knows where the food is, but they all know how far they are from the food. Thus, the most straightforward and efficient strategy is to search the area around the bird who is nearest to the food. After one search iteration, the birds combine all their information. The flock will use the combined information for the next search iteration. PSO shows has its advantages in its simple implementation, high precision, and fast convergence. More importantly, it shows superiority when it is used to solve practical problems. PSO can be coded to run in parallel, which makes it suitable to deal with an actual problem. Sometimes PSO is modified to search the neighbours of a part of the whole flock. This idea accelerates the process to find the global optima. However, it may increase the scale of the algorithm’s program. Owing to these strengths, PSO has attracted increasing attention in recent years [45, 46, 47].

In our implementation, the algorithm moves through possible solutions (permutations) within the search space. PSO finds the best solution by following both the temporary best solution and the global best solution. As with SA, PSO starts searching from a randomly generated initial solution. PSO uses a manipulation similar to “cross-over” in GA. In the first round, each solution that is not the temporary best performs “cross-over” with the temporary best solution to “produce” new solutions. PSO compares all the fitness values of all the new solutions to find the new temporary best solution. After finishing the first round, all the newly produced solutions perform another round of “cross-over” with the global best solution and perform the same operation as in the first round. The new temporary best solution and new global best solution are saved, and the new temporary best fitness value and global best fitness value are updated. In addition to these two operations, to avoid the algorithm being trapped in local optima, we added a "mutation" operation in the implementation. In each iteration, one bird in the whole flock is chosen to mutate. It performs operation 2-OPT to generate the new solution. This operation simulates the situation that a bird occasionally loses its way but finds a great treasure. Similar to the other heuristic algorithms, the program stops when it evolves a prefixed number of times, or the best fitness value remains unchanged in several continuous evolutions. Compared with GA, PSO's running time is relatively short. Thus, we dismiss the second stopping criterion as we did in SA.

In our experiment, we set the parameters of PSO as follows. The flock was set to have a population of 100 birds. The maximum evolution time was set to be 1000. Different from GA, PSO was forced to "mutate" once in every iteration. Thus, no mutation ratio was given here.

3.2.4 Tabu Search

Glover first presented TS in 1986 [48]. TS is an extension of the local search algorithm and one of the meta-heuristic random searching approaches that simulates the process of human intelligence. Compared with GA and PSO, TS abandons the idea of producing the new solution from two parent solutions. Instead, it searches the whole search space but avoids the places around the temporary optima. This is different from the traditional idea that we should keep searching the area around the temporary optima to obtain a better solution. TS has drawn attention from the research community in recent years [49, 50] because it produces good performances in many research areas, especially in bi-criteria problems [51, 52]. Because MWLP is exactly a problem with two constraints (vertex weight and edge length), we tested TS to see if it performs well with MWLP.

TS tries to find the global optima by avoiding searching in the direction that a guaranteed local optima exists. The algorithm uses a data structure called a “tabu list” to structure the search process, and help jump out of local optima. The local optima solution is stored in the list. The algorithm will avoid searching the solutions in the “tabu list”, and the neighbours of the elements in the list. A number called the “tabu length” limits the size of the “tabu list” is introduced into the algorithm to make it release some searching area after new local optima solutions are found. The “tabu list” is managed as a first in first out (FIFO) queue. The “aspiration criterion” is another unique element in TS. If a solution is found during one iteration and it is the result “best so far”, it will not be put into the “tabu list” and so its neighbour would be considered. The three data structures introduced above establish the primary structure of TS and distinguish it from the other heuristics. In addition to these, “diversification” and “intensification” are the two most significant manipulations in TS. “Diversification” is a manipulation that forces the algorithm to search the area which has not been searched. The opposite manipulation, “Intensification”, forces the algorithm to search an area near the temporary optima to find a better result. These two manipulations help the algorithm to further jump out of local optima while still finding the best solution. Sometimes a parameter called “frequency” is used to extend the searching space. It limits the time for which a solution can be put into the “tabu list”. Usually, a guaranteed local optima should be put into the “tabu list” as long as it is not in the list regardless whether or how many times it has been a “tabu”. Namely, by default, “frequency” is set to be unlimited. It is considered as inefficient because the multiple appearances of one solution indicate that the algorithm has already been trapped. “Frequency” is introduced to keep the solution with multiple appearances out of the “tabu list” so as to jump out of the local optima.

In our experiment, we dismissed the parameter “frequency” to make the algorithm run as freely as possible. 2-OPT was performed between visiting sequences of two vertices in one solution to evolve the solutions. The tabu length was set to $\sqrt{\frac{\text{Vertex number}^2}{2}}$ and 100 threads were active in the search space at the same time. The threads stopped searching if the threads finished a particular number of iterations. As in GA, this variable was chosen to be 50.

3.2.5 Ant Colony Optimization

Dorigo *et al.* initially proposed ACO in 1991 [53]. This method simulates an ant colony searching for food. If one ant finds the food, it goes back to the colony and leaves a pheromone trail on its path from the food back to the colony. The other ants follow the pheromone to find the best route. It is a very powerful heuristic algorithm with good robustness, acceptable running time, and scale for the application program. ACO can easily be coded to run in parallel. With these advantages, ACO has become very popular [54, 55, 56].

ACO mimics the method that ants use to find the shortest route between their colony and the food sources. Ants find the route by leaving a volatile pheromone on their way to the food source and back from the food resource to their colony. When an ant needs to find a way to go, it chooses the route with a higher pheromone concentration. Correspondingly, the pheromone concentration over a path becomes higher when more ants go through it. This leads to the result that more ants tend to choose it. When the ants change their colony location, this positive feedback mechanism also takes place and helps the ants quickly find their new colony site. If a route is not used frequently, the pheromone deposit will weaken, so the ants will tend to abandon it. When no pheromone has been found, a single ant will randomly choose any possible path if no other factor affects its choice. This mechanism assures the ants can find the shortest route even if several individuals become lost. ACO helps a single ant employ the collective intelligence of the whole colony.

I designed an approach different from the mechanisms in GA or TS to generate new solutions for ACO. This new ACO does not use 2-OPT to create its new solutions. It obtains the solutions by choosing the next vertex on an individual basis. Every ant first randomly selects a vertex as its starting vertex. Then the ant fetches the pheromone values of every path from a table maintained by the system. The ant computes the importance of all the unchosen paths by adding the path's pheromone value and the fitness value of the permutation if the path is chosen with a ratio of 1:5. After the fitness value of every unchosen path is computed, the ant decides its next vertex by performing roulette wheel selection. The ants keep doing this until all the vertices are chosen. The system has a population of 50 ants. Every edge's pheromone concentration is multiplied by 0.9 after each iteration. To fully understand the capability of ACO, we designed two mechanisms to update the pheromone value after each iteration. The first is that the algorithm updates its pheromone table by only using the best solution. The second is that the algorithm updates its pheromone table by using every solution. Only the second mechanism is used in the comparison on behalf of ACO in this chapter because the first mechanism is not as good as the second. However, this shows its capacity when the ACO is accompanied with VNS to solve MWLP. We provide a brief comparison of these two mechanisms later in this chapter. More details of the application of the first ACO mechanism are given in the next chapter. The program does the same thing when performing the update. It just multiplies the old pheromone value by the volatilization parameter 0.9 and then adds the reciprocal of the solution's fitness value. Here the numerator 1 (the reciprocal can be seen as division with numerator 1) is carefully chosen from many candidates. I tried using a parabola going upwards, parabola going downwards, hyperbola going upwards, hyperbola going downwards, linear going upwards, and linear going downwards. None of these choices gave better results than using a constant, and they had the drawback that they consumed more time.

3.3 Experimental Results and Conclusions

We examined the efficiency and effectiveness of heuristic approaches for MWLP. The DP algorithm proposed by Wu was used for some instances' comparisons. In this section, some analysis of the choice of heuristics' initial solutions is first given. Then the experimental settings and the test results are introduced. A comparison of using different heuristics to solve different scale MWLP completes the results.

First, the terms used in this section should be clearly introduced. "Instance" is used to describe different MWLP problems. MWLP problems with different graphs or different vertex weight distributions are considered as different "instances". "Experiment" is used to describe the whole comparison process or a whole process of using any heuristic to solve an instance. It may have different numbers of iterations. "Test" is used to describe a set of experiments running on an instance. The outputs of one test are used as a whole to measure a heuristic's search capacity.

3.3.1 Initial Solutions

The choice of initial solutions of the heuristics is important. It determines the start point from which the heuristic begins to search. A good choice of the initial solution can help the heuristic to not only improve its probability of finding the global optimum, but also reduce the time it consumes. However, compared with a randomly chosen solution, an intentionally selected initial solution has its own shortcomings. It could limit the heuristic's search space, which may trap the algorithm into a local optimum. In addition, the solution may need a series of delicate computations to calculate. The computations can be very time consuming and reduce the heuristic's efficiency.

An experiment was implemented to test what kind of initial solutions should be chosen in the heuristics for MWLP. Our experiment tested the effectiveness of two groups of heuristics when solving an MWLP with 20 vertices. Both groups contained all five heuristics introduced above. One group's initial solutions were allocated randomly, the other group's were generated by a greedy algorithm. The greedy algorithm was run by a mechanism that the vertex with higher weight is chosen earlier. All the heuristics shared the same number of iterations, 100, and every heuristic was run 100 times to avoid coincidence. Except for SA and TS, every heuristic was optimized from multiple initial solutions. In this situation, if the algorithm was in the second group, one of its initial solutions was generated by the greedy algorithm, and the rest were all generated randomly. Because all the heuristics had the mechanism to save every iteration's best result, even one solution was enough to affect the final results. The results of the experiment are given in the following.

Heuristics:	ACO	GA	PSO	SA	TS
Best Result:	1104.810	1104.810	1104.810	1104.810	1104.810
Average Result:	1109.885	1209.025	1335.477	1305.334	1304.765
Worst Result:	1127.374	1417.875	1595.277	1834.458	1710.599

Table 3.1: Results of the group with randomly generated initial solutions.

Heuristics:	ACO	GA	PSO	SA	TS
Best Result:	1104.810	1104.810	1104.810	1104.810	1347.764
Average Result:	1109.399	1144.565	1177.038	1292.284	1362.258
Worst Result:	1127.374	1273.265	1363.909	1705.343	1362.554

Table 3.2: Results of the group with intentional selected initial solutions.

Tables 3.1 and 3.2 list the results of the two groups of heuristics. The best result, average result, and worst result illustrate the best, average, and worst value of the 100 outputs for every test. Most heuristics made relatively small improvement when their initial solutions were selected by the greedy algorithm. Two exceptions were PSO and TS. For PSO, the average value and worst value improved drastically. Nevertheless, both groups of PSO found the global optimum 1104.810 (we knew this from the output of the DP algorithm). In terms of TS, it verified the hypothesis we mentioned above. The selected initial solution trapped the algorithm into a local optimum. In general, a selected initial solution can help the heuristics slightly improve their search capacity, sometimes the improvement can be great. However, this mechanism may also cause unexpected results.

For the determination of the heuristics' initial solutions, considering the efficiency, methods more sophisticated than the greedy algorithm were passed. Taking every factor into account, the greedy algorithm was chosen as the mechanism to compute the initial solutions for the heuristics.

3.3.2 Experimental Settings

The heuristic algorithms were tested in the experimental environment introduced in Chapter 1. The tests consisted of MWLP instances with 12, 16, 20, 30, 40, 50, and 100 vertices. GA, SA, PSO, TS, and ACO were conducted for all the instances. DP proposed by Wu was used as a comparison for the cases in which the number of vertices is 12, 16, and 20. It took about one full week for the small-scale server to run the DP program to solve the instances with 20 vertices. The DP algorithm's running time increased exponentially with the number of vertices, which was as expected. In practice, more I/O manipulations also affected the running time.

We tried the DP code on an instance with 22 vertices. The program kept running for one whole month without producing any result. As a result, I gave up running the DP for the instances with more than 20 vertices. In each set of experiments, all five heuristic algorithms were tested by performing 30, 50, and 100 iterations, respectively.

Table 3.3 shows the results and running time for DP algorithm when solving different MWLP

Vertex Number:	12	16	20 (uni_weight)	20 (ran_weight)
Results:	593.3672	777.4809	1.1048e+3	1.5282e+3
Running Time:	4.05s	849.59s	535384s	536725s

Table 3.3: Results and running time of the DP algorithm for multiple MWLP instances.

instances. As explained previously, the numbers of instances of vertices were limited to be at most 20. As for the instances with 20 vertices, two cases were tested. The weight values of the vertices of one of the two cases were distributed uniformly. In this case, the weights of the vertices were constrained to 0.2, 1/3, 0.5 ,and 1. These four weight choices were generated from a wireless sensor network (WSN) simulation. The weight of a vertex means the number of messages it produces in every minute. This case is represented by 20 (uni_weight) in Table 3.3. In the other case of 20 vertices, the weights were generated randomly within the interval of $(0, 1]$. This case is represented by 20 (ran_weight) in the table. The experimental results have up to four digits after the decimal point. Regarding the running time, the values have two digits after the decimal point when the number of vertices is 12 and 16. As for the two instances with 20 vertices, the running times were very long. It is meaningless to record the instances' exact running times. The difference between the two cases with 20 vertices' running times is about 0.25%.

The results of heuristic approaches for MWLP instances with 12, 16, 20, 30, 40, 50, and 100 vertices are shown in Figures 3.1–3.8, respectively. The figures illustrate the average value, maximum value, minimum value, average time, and standard deviation of different cases. Time is in seconds. To avoid the coincidence, for every number of iterations and every number of vertices, we executed 100 experiments. The "average value" shows the average objective function values obtained by the 100 experiments. The "maximum value" represents the maximal objective function values found in 100 experiments. They are the worst cases of all these 100 experiments. On the other hand, the "minimum value" represents the minimum objective function values in 100 experiments. Obviously, they are the best cases of the experiments. The “average time” shows the average execution time for the 100 experiments. Thus, if we take all 100 experiments as a whole, the full test time can be simply computed by multiplying by 100.

3.3.3 Experimental Results

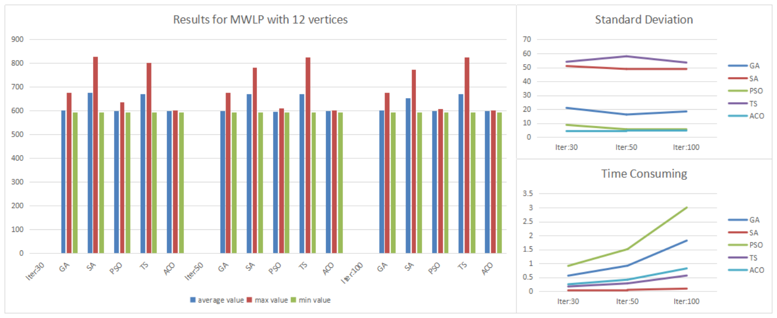

Figure 3.1: Experimental results for the instance with 12 vertices.

Figure 3.1 shows the results for five heuristic algorithms solving MWLP with 12 vertices. As shown in Table 3.1, the global optima were known from the result of the DP algorithm. As shown in the illustration, all five heuristics found the global optima regardless of the number of iterations. As the number of iterations increased, the values of the heuristics’ worst outputs, along with the outputs’ average values, decreased slightly. Taking PSO as an example, when the number of iterations was 30, the worst result was 6.37e+2. When the number of iterations reached 50, the value was 6.11e+2. Finally, when the number of iterations was 100, the worst result improved to 6.08e+2. For the average results, the values were 6e+2, 5.98e+2, and 5.99e+2, respectively, as the number of iterations increased. Though the average value fluctuated, the worst value kept a trend of decreasing with increasing numbers of iterations. The unit for running time is seconds. The SA algorithm consumed the least time regardless of the number of vertices, which is due to the relatively simple architecture of SA. The time consumption of GA and PSO were quite high.

Though their efficiencies may seem low, their effects were good, especially PSO in this case. As for the effectiveness, ACO and PSO ranked the best among the five methods. The standard deviation value shows they were also the most stable methods. By integrating all the evaluation criterion, ACO ranks first among the five methods.

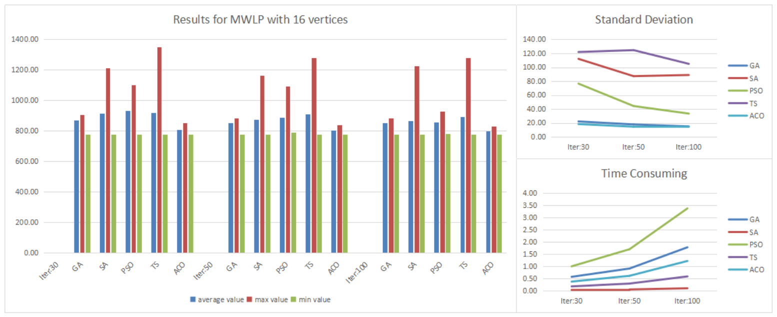

Figure 3.2: Experimental results for the instance with 16 vertices.

Figure 3.2 shows the experimental results of MWLP with 16 vertices. The global optimum was also known from the result of the DP algorithm. Different from the instance in which the number of vertices was 12, all 15 tests did not find the global optimum. Ironically, the five cases that missed the global optimum are ran by PSO and ACO. Whereas they performed most effectively in the 12 vertices instances. Among all the six tests run by these two methods, only PSO iterated 30 times found the global optimum once. This result is surprising and difficult to explain. As mentioned above, every test ran 100 small cases, which should be enough to rule out the chance of coincidence. Through carefully studying the six tests of PSO and ACO algorithms, differences can be found. Though ACO did not find the global optimum, its average values led the pack of the heuristics. Furthermore, it found the value 7.793238e+02 (only 0.25% worse than the global optimum) many times in all the three tests. This value is obviously a local optimum in which the algorithm was trapped. For PSO, it did not just miss the global optimum, it also performed worse in general compared with its performance when the number of vertices was smaller. The major sign of that was its three results of average values were larger than those of GA. In conclusion, ACO was still effective though it was trapped in a local optimum. On the flip side, PSO performed badly for this instance. PSO may only be capable of solving MWLP with a very small number of vertices. In terms of the effectiveness of the other algorithms, GA performed better than it did in the previous case. TS and SA performed poorly no matter how many iterations they ran. The average values and maximum values of their outputs were too large. From the viewpoint of stability, the standard deviation values of the outputs of TS and SA were large, which means their performances were highly unstable. The standard deviation values of the outputs of GA and ACO indicated their high stabilities. From the viewpoint of efficiency, SA and TS consumed the shortest time. However, from the viewpoint of effectiveness, they are not good choices in this case. In general, ACO and GA ranked highest in this case if we do not consider their time consumption.

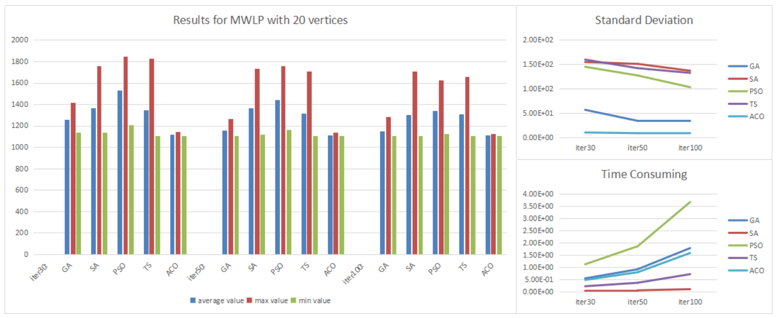

Figure 3.3: Experimental results for the instance with 20 vertices with uniform weighted distribution.

Figure 3.3 shows the experimental results of MWLP with 20 vertices. In this case, the values of the weights of vertices were distributed uniformly. This is considered as a small-scale MWLP. The global optimum was known from the output of the DP algorithm. It was found by heuristics in 9 of the 15 tests. Even though the global optimum was found in more than half of the cases, PSO did not obtain it even once. All the other heuristics (GA, SA, TS, and ACO), found the global optimum in the tests with 100 iterations. Out of the 100 experiments, they had 18, 1, 3 and 22 hits, respectively. The number of hits means the number of times a heuristic found the global optimum. Regarding the tests where the number of iterations was 50, GA, TS, and ACO found the global optimum. Out of the 100 experiments, they found it 8, 2, and 17 times, respectively. For the last case, tests with 30 iterations, only TS and ACO found the global optimum and their hit numbers were two and five, respectively. Based on these results, the effectiveness of ACO is obviously better than the other heuristics. Considering the stability, the values of standard deviation of the outputs of ACO were the smallest compared with the other heuristics when they had the same number of iterations. Thus, ACO demonstrated the highest stability among the heuristics for this instance. In conclusion, from the viewpoint of effectiveness, ACO performed the best. For the efficiency, it took the server 5.35e+05s to find the global optimum when it ran the DP method. In contrast, ACO took only 1.58 s for one experiment when the number of iterations was 100. SA has a remarkably low time consumption of only 9.90e-02 seconds. Compared with the DP method, all five heuristic algorithms had relatively low time consumption. PSO's running time was the longest in this situation. It took almost 4 seconds to finish one experiment. The total time consumption for 100 experiments was a little more than 6 minutes. Nevertheless, this running time is still acceptable when the situation is not urgent.

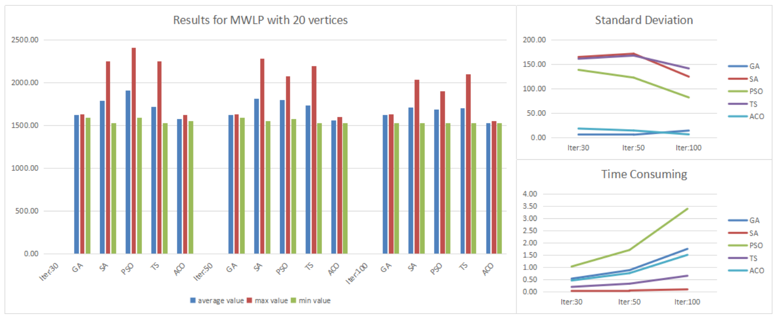

Figure 3.4: Experimental results for the instance with 20 vertices with randomly weighted distribution.

Figure 3.4 shows the experimental results of MWLP with 20 vertices. Different from the case above, here the weight values of the vertices were distributed randomly. We also obtained this instance's global optimum by running the DP algorithm. The global optimum was only found by heuristics in 5 out of the 15 tests compared with its counterpart where the number was 9 out of 15. Among the five tests in which the global optimum was found, two of them were ran by SA and the rest were ran by TS. These two heuristics performed the worst during the past tests from the effectiveness side. The numbers of iterations of the tests in which the global optimum was found were 30 and 100 for SA. All three tests of TS found the global optimum. However, the hit numbers were 1, 1, 2, 2, and 3, respectively. For the tests containing 100 small experiments, they were just small probability events. These tests only presented SA and TS may have the capability of jumping out of the local optima, but no guarantee of higher searching capacities compared with other three heuristics. The average values of their outputs proved this from another side. Their average values were around 17% and 18% worse than that of ACO when their numbers of iterations were all 100. Considering ACO, though it missed the global optima, ACO found the result 1.53e+3 many times. It is only 0.2% worse than the global optima. This is the same as what happened in the instance with 16 vertices. ACO was again trapped in local optima. It represented a shortcoming of ACO that it has a tendency to be trapped in local optima. Compared with the instance above, all the five heuristics performed almost the same from both effectiveness and efficiency sides. GA showed more stability, but only very slightly. Nevertheless, the heuristics hit much fewer times in this case, which showed that heuristics are much more suitable for solving the MWLP problem with a limited selection of vertex weight values.

Figure 3.5 shows the experimental results of MWLP with 30 vertices. It is considered as the first medium-scale MWLP problem in our experiments. As we explained above, DP algorithm

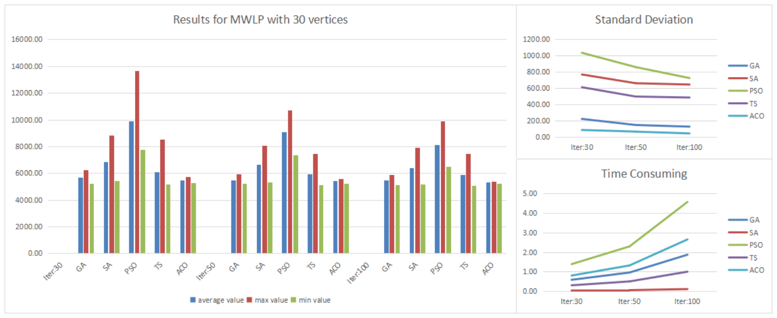

Figure 3.5: Experimental results for the instance with 30 vertices.

consumes too much time when the problem has more than 20 vertices. This made us skip the DP implementation for this instance and so we did not know the instance's global optimum. From the viewpoint of the effectiveness, ACO performed the best regardless of the number of iterations. In addition, it demonstrated incredible stability. Its standard deviation values were the smallest among the five tested heuristics regardless of the number of iterations. In addition, as the number of iterations increased, ACO's standard deviation value curve almost became horizontal, which implied its stability even for fewer iterations. Different from ACO, the other four heuristic methods' standard deviation value curves went down significantly as the number of iterations increased. That is because, with the increase in the number of iterations, these heuristics tended to reach the upper limit of their searching capacity. The highly stable performance of ACO can also be a disadvantage. It may imply that the algorithm tends to keep searching a small range of candidates. In other words, it is more easily trapped in local optima. Evidence has been found in other instances as well. Regarding the efficiency side, PSO performed poorly as also demonstrated the previous several tests. It consumed the longest time while outputting the worst results. SA consumed far less time compared with the other heuristics. However, considering the real time, none of the other four heuristics consumed more than 5 seconds in one experiment even when the number of iterations was 100. As for the tests with 30 iterations, the differences in time consumption between the four heuristics except SA were no more than 1.5 seconds. Thus, none of them performed much better or much worse than the others. All in all, for most situations of the real-world applications, the efficiencies of all five heuristics are acceptable.

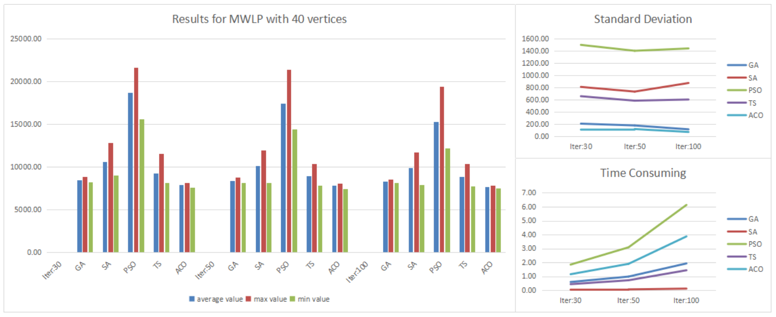

Figure 3.6: Experimental results for the instance with 40 vertices.

Figure 3.6 shows the experimental results of MWLP with 40 vertices. In general, the dominant algorithm was also ACO in this case. GA ranked second and performed almost as well as ACO. With regards to effectiveness, ACO performed the best among the five heuristic methods. Following ACO, the next heuristics were GA, TS, and SA. PSO performed the worst, and is not a good choice for this MWLP instance. From the viewpoint of stability for these methods, the standard deviation value curve of SA was interesting. It decreased slightly as the number of iterations increased from 30 to 50, but increased as the number of iterations increased from 50 to 100. The fluctuation implied that the setting of the number of iterations might not fit this instance. As the problem size increased, only using 100 iterations may not be sufficient for SA to demonstrate its abilities. On the other hand, it consumed the least time among the tested heuristics. SA again ranked first for efficiency. Combined with the analysis in the effectiveness part, a larger number of iterations should be tested to find the limit of AC's search capacity. More details are given in the analysis section of this chapter. ACO and GA output the best results, and although ACO consumed twice as much time as GA, the real time consumptions were still acceptable (4 compared with 2 seconds).

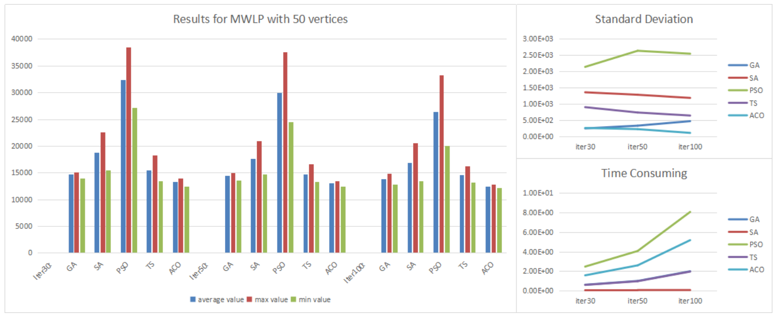

Figure 3.7: Experimental results for the instance with 50 vertices.

Figure 3.7 shows the experimental results of MWLP with 50 vertices. This was the medium-scale MWLP problem we tested. For effectiveness, unsurprisingly ACO again performed best. As the number of vertices increased, ACO cemented its dominant position. Its worst outputs were even better than the best outputs of SA and PSO. Meanwhile, it also found the best solutions out of all the heuristics and remained highly stable. GA ranked second. Its outputs were only a little worse than those of ACO, but much better than those of the other heuristics. Among the other three methods, TS began to show advantages over SA. When the problem scale was smaller, SA and TS showed similar search capacities. However, with the expanding problem size, TS output better results. Regarding efficiency, ACO performed second worst of the five heuristics. On the other hand, GA ranked the third and almost the same as the second place heuristic (TS). It showed a good balance between its efficiency and effectiveness. However, it is based on a premise that the heuristics are compared with the same number of iterations. If the 15 tests were combined, the conclusion changed. The test in which ACO had 30 iterations produced the worst outputs but consumed the least time among the three ACO tests. The test where GA had 100 iterations produced the best outputs but consumed the longest time among the three ACO tests. However, the former test produced better outputs and consumed less time than the latter. In conclusion, for this instance, ACO not only produced the best results but also found a good balance between effectiveness and efficiency.

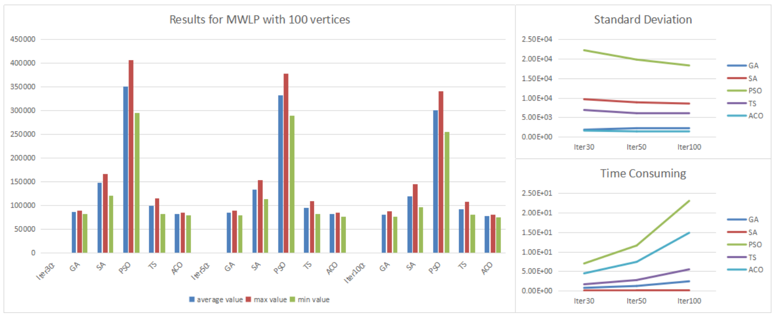

Figure 3.8: Experimental results for the instance with 100 vertices.

Figure 3.8 shows the experimental results of MWLP with 100 vertices. This was the only large-scale MWLP problem we tested. From both the effectiveness and efficiency viewpoints, P-SO ranked the worst for this instance. SA consumed the least time, but its outputs were only better than those of PSO. The performance of TS was exactly in the middle place (considering both effectiveness and efficiency). GA and ACO both produced good results and showed amazing stabilities regardless of the number of iterations. Though ACO showed some advantages in effectiveness over GA, it consumed much more time than GA. The choice of these two heuristics should be based on the actual demand. If the output is required to be as good as possible but the time consumption is trivial, ACO should be chosen in this situation. On the other hand, if the application is highly time sensitive and does not require the best result, GA is the better choice.

3.3.4 Conclusions

The experimental results proved that all the five tested classic heuristics can be implemented to solve MWLP. However, different heuristics have different advantages and disadvantages, and they demonstrate different features when the problem size changes.

For small-scale MWLP problems, the time consumption of all five heuristics consumed was relatively low. ACO, GA, and PSO were the three heuristics which showed the best searching capacities.

Regarding the medium-scale MWLP problems, SA and TS ran fastest. GA and ACO were in the second tier. PSO consumed the most time. From viewpoint of effectiveness, GA and ACO produced the best results. Their outputs were far superior to the outputs of the other three heuristics. The most suitable method for medium-scale MWLP is ACO, as it struck a good balance between efficiency and effectiveness.

For large-scale MWLP problems, PSO performed poorly. It consumed the most time and output the worst results. Compared with PSO, the output of the other four heuristics were almost on the same level, although ACO and GA were slightly superior. As the problem size expanded, the time consumption of ACO was much higher compared with the other heuristics. Considering both effectiveness and efficiency, the most suitable method for a large-scale MWLP is GA.

3.4 Theoretical Analysis

In this section, a short theoretical analysis of the efficiency and effectiveness of the heuristics represented in the experiment results is given. In addition, as mentioned in the previous section, the test result and analysis of using the SA algorithm with a larger number of iterations to solve MWLP are given.

3.4.1 Test for SA Methods with Larger Numbers of Iterations

As presented in the previous section, regardless of the number of vertices in the MWLP instances, the time consumptions of their SA algorithms were relatively small even when the number of iterations of the algorithms was increased to 100. In this section, the number of iterations of SA was set to 500. Considering that all the heuristics' running times were relatively short and they all produced good outputs in small-scale problems, two medium-scale instances and a large-scale instance were chosen in this experiment. They are the instances with 30, 50, and 100 vertices, respectively. The other heuristics' algorithms with 100 iterations were chosen for comparisons. The results are shown in Figures 3.9–3.11.

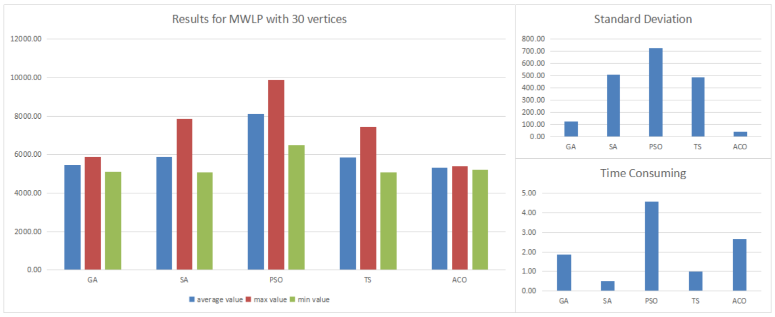

Figure 3.9: Experimental results for the instance with 30 vertices for the SA method with 500 iterations.

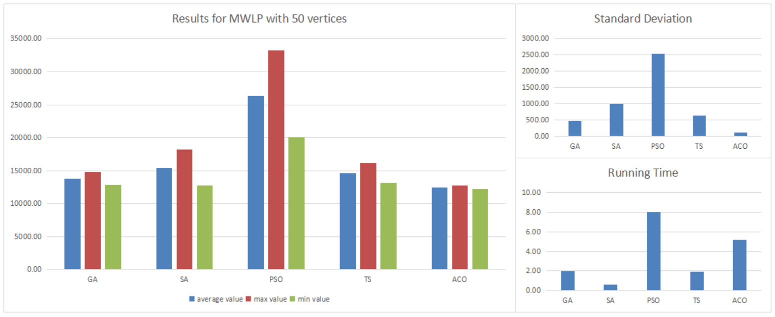

Figure 3.10: Experimental results for the instance with 50 vertices for the SA method with 500 iterations.

From the figures, it can be seen that the SA method maintained its advantage of a relatively short running time. With a greater number of iterations, SA produced better outputs. Compared with the experimental results shown in Figures 3.5, 3.7, and 3.8, the stability of the SA algorithm also increased. Such small values in standard deviation of the outputs showed that the algorithm

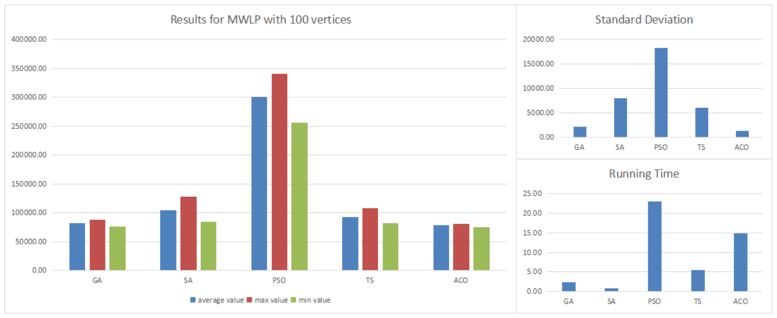

Figure 3.11: Experimental results for the instance with 100 vertices for the SA method with 500 iterations.

had reached the limit of its capacity to search the search space to some extent. However, it only produced an improvement compared with itself with fewer iterations: it did not produce a more powerful searching capacity compared with the TS algorithm, let alone GA or ACO.

3.4.2 Efficiency Analysis

From the analysis of results in the previous section, ACO and GA are the two most effective algorithms. Thus, here we try to analyse the time-consumption patterns of ACO and GA. Figure 3.12 illustrates the time consumed by ACO and GA when the number of vertices was 12, 16, 20, 30, 50, and 100. We dismissed the instance with 20 randomly distributed weighted vertices and the instance with 40 vertices in this analysis to make the illustration more readable. Different lines represent different numbers of iterations as shown in the figure.

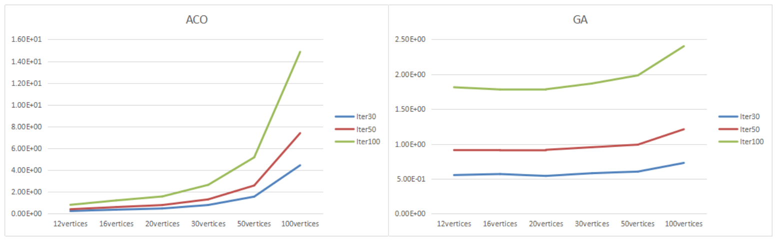

Figure 3.12: Time consumed by ACO and GA.

For different numbers of iterations, the time-consumption curves were similar for both heuristics. We use the instances with 100 iterations as examples. Linear fitting was performed for the time-consumption curves of both algorithms. The results are shown in Figure 3.13. The R^2 values were 0.1262 and 0.1516, respectively, which implies that the time-consumption curves of both algorithms were non-linear.

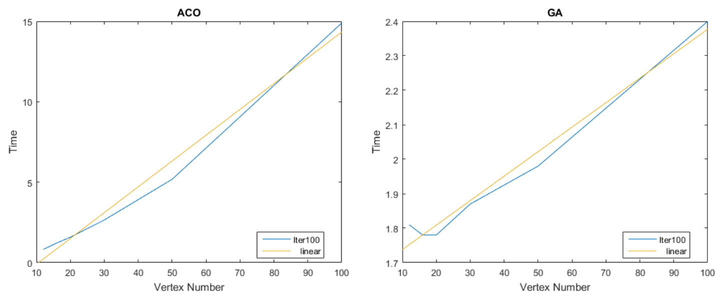

Figure 3.13: Linear fitting.

ACO's time consumption increased steadily as the number of vertices of the instances increased at the beginning. When the number of vertices increased from 50 to 100, it increased drastically. This may have been caused by ACO's central mechanism. As the number of vertices increases, the size of ACO's pheromone table increases exponentially. The cost of maintaining and checking it would increase quickly.

In terms of GA, different from ACO, its running time remained low until the number of vertices increased to 50. When the number reached 100, its running time increased slightly, but not as much as that of ACO. The explanation for this phenomenon can also be taken from its algorithm structure. The key manipulations of GA are "cross-over" and "mutation", whereas their costs are almost independent of the number of vertices. This explains why the time consumption of GA was relatively low when the problem size increased compared with ACO.

3.4.3 Effectiveness Analysis

In our tests, ACO obtained the best results. We believe this is because the inner structure of ACO fits MWLP well.

In the process of solving MWLP, both the weight of vertices and the distance between vertices affect the result. The vertices with higher weights and the edges with shorter distances tend to be visited earlier. The primary optimization objective is to find a balance between higher weights and shorter distances. That implies the global concept of MWLP is more important than other NP-hard problems (such as TSP).

The ACO algorithm we used in the experiment updates its pheromone data globally. After any iteration, the algorithm uses a certain number of the global best results it finds in that iteration to update its pheromone table. This means that all the vertices are treated as a whole when they are updated or chosen, which is very different from the other algorithms. Take GA as an example. It uses "cross-over" and "mutation" manipulations to look for new results. However, both update the result by changing the position of only two or three vertices in one iteration. It deals with only local vertices compared with ACO.

To make this more credible, we designed an ACO that updates its pheromone table during every step, but not globally. The new ACO was tested with the instance with 50 vertices and 100 iterations. The results are shown in Figure 3.14, in which AC_Local denotes the new ACO algorithm. This algorithm and its output will also be used in the next chapter as a comparison with the existing ACO algorithm. They will be denoted as AC1 and AC2 in the next chapter respectively. When the update method of the pheromone table was changed, the effectiveness of ACO decreased.

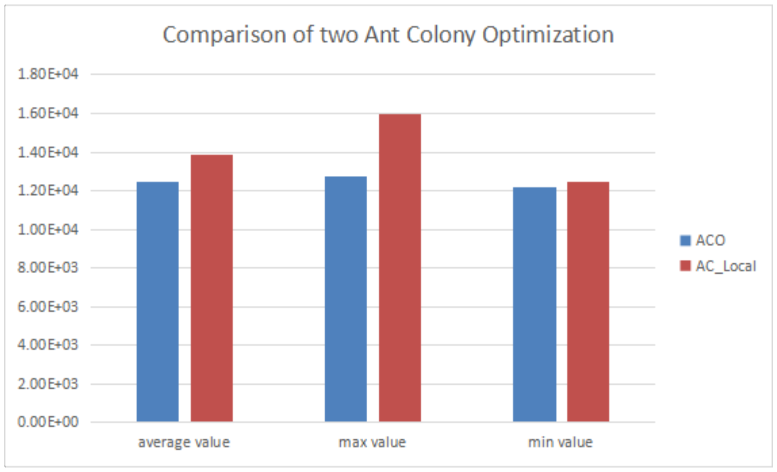

Figure 3.14: Comparison of results for two ACO algorithms.

3.5 Conclusions

In this chapter, we used a variety of heuristic algorithms to solve WMWP. Problems with different numbers of vertices were tested using five classic heuristics. For all instances, the heuristics used 30, 50, and 100 iterations. The experimental results revealed that the tested classic heuristic algorithms (GA, SA, PSO, TS, and ACO) can find feasible solutions for MWLP.

Through experiments and analyses, ACO was proven to be the most appropriate heuristic algorithm for small-scale and medium-scale MWLP instances. In these two situations, ACO is verified as able to find the best solution in the shortest time compared with the other heuristics. For the large-scale MWLP problem, ACO could find the best result if it has the same number of iterations as the other heuristics. However, the running time of ACO was too long compared with the other algorithms. In this case, GA demonstrated a good balance between effectiveness and efficiency.

In the experiments above, all the heuristics were run sequentially. It was difficult to code GA, SA, and TS in parallel because of the limits of their inner structure. However, PSO and ACO could be coded in parallel with few modifications. This could be a direction for the future work to improve the efficiency of ACO and PSO, especially for large-scale MWLPs.

4 New Heuristic Algorithms

In this section, we introduce two new algorithms for MWLP. Similarly to the classic heuristics presented in the previous chapter, the algorithms proposed here are also designed as evolutionary algorithms. However, by combining the main ideas of multiple heuristics, the newly proposed algorithms are more complex and capable of finding better solutions for MWLP. The algorithms proposed here are based on P system and VND, respectively. Owing to the algorithms' huge scales, they consume significant amounts of time if they are designed to run sequentially. To avoid the enormous time consumption, we developed mechanisms to run both algorithms in parallel.

In the beginning of this chapter, we introduce the main idea of P system as well as the method for MWLP based on it. The parallel mechanism is also given. We then repeat this for VND. Next, experimental results are presented to illustrate the effectiveness of the algorithms. Finally, we provide an analysis from the perspective of the effectiveness and efficiency of the algorithms.

4.1 The algorithm based on P system

P system (also known as membrane computing) is a computation model proposed by Păun in 1998 [57]. Păun is a computer scientist whose study area is biological computing. Inspired by the structure of biological cells, he proposed the computational model by simulating the process that a cell uses to handle compounds in a circulation system consisting of multi-layer membranes.

P system has drawn attention since Păun proposed it. Researchers have designed variants of the P system. Many of them have been proved to have computation capacity equal to a Turing machine. Inspired by the biochemical reaction in live cells and the transportation mechanism of biochemical substances between cells, Nishida proposed the "membrane algorithm", an evolutionary algorithm based on the idea of the P system for combinational optimization problems [58, 59]. Nishida proved that the algorithm performs better than conventional evolutionary algorithms when solving TSP. In addition to TSP, algorithms based on P system have also been designed to solve other NPC problems, such as SAT [60] and Hamiltonian tour [61]. All these algorithms are proven to be both efficient and effective. Moreover, P system has been applied in many inter-disciplinary situations. Algorithms based on it have been designed and implemented to help solving problems in systems biology [62, 63, 64], computer graphics [65, 66], semantics [67, 68, 69], economics [70, 71, 72], and ecology [73, 74, 75]. For real-world applications, P system has also been used to solve problems in radar signal processing [76], minimum storage problems [77], DNA sequence design [78], cooperative driving schedules [79], and optimization of industrial processes [80]. More over, it is even used in decryption [81].

Recently, an increasing number of meta-heuristic algorithms based on P system have been proposed. For instance, in 2014, Niu *et al.* [82] proposed a meta-heuristic algorithm combining the structure of P system and the idea behind ACO to solve the capacitated vehicle routing problem (cVRP). In the same year, another meta-heuristic algorithm, also based on the structure of P system, was presented by Yan *et al.* [83]. They used SA to communicate between different membranes to solve the weighted optimization problem for case-based reasoning. In 2017, Dong *et al.* [84] proposed a multi-objective evolutionary algorithm based on a tissue P system to solve the vehicle routing problem with time windows (VRPTW). The evolutionary algorithm combined the discrete glowworm evolution mechanism (DGEM) and variable neighbourhood evolution mechanism (VNEM).

The three kinds of P system studied most frequently are cell-like, tissue-like, and neural-like. In this section, we propose an algorithm based on the structure of a cell-like membrane system to solve MWLP. We briefly introduce the P system and define the concept of a "unit operation", then go into the details of the proposed composite algorithm.

4.1.1 Introduction to the P system

Similarly to other natural computation models, P system is also inspired by an existing process in the natural environment. It simulates the chemical processes between different layers of the membranes within a cell. During the evolution process, the cell is considered as a frame. The chemical compounds within it are the subjects that are involved in the evolutionary process. When passing through a membrane, the compounds are involved in some chemical process with other compounds or some pre-stored catalyst inside the membrane. Through iterations of this process, new compounds are produced and then involved in the new process. The final output will be produced after some stopping criterion is met.

Figure 4.1 illustrates the structure of a cell. Inside it, four colours represent different region-s. The boundaries between them are membranes. During the evolutionary process, they work as channels. The compounds move through them and react chemically with each other or self-react if a catalyst exists. According to the functions they provide, various membranes contain different cat-alysts. They cooperate with each other to make the whole system work. Imagine a situation where a compound passes through the skin from the outside environment. The cell needs to decompose it, absorb the nutrition, and then eliminate the waste. A sequence of chemical reactions needs to occur in different regions to accomplish these tasks. The substances can be either a reactant or a product inside or outside of a membrane. Multiple rounds of chemical reactions may happen in different regions of the cell before it finally produces the complex reaction product.

Chemical reactions are very similar to mathematical calculations. They both have inputs and outputs. In different environments, the outputs can be different even though they have the same inputs. If we consider the chemical reactions inside the cell from this aspect, considering them as computations, the whole process is very likely a computational model. The initial data passes through different membranes to “evolve” itself. At the end of the computation, the model outputs the final result as the cell produces its final product.

Păun first proposed this as a model for a biological computing. Several years later, some simulators were designed by different organizations. Nevertheless, it is still not a computational model that fits electronic computers well. The main issue is that it runs calculations in parallel in different regions whereas the ongoing jobs in an electronic computer are inherently sequential. Many DNA-computing scientists allege they can solve NP-complete problems in polynomial time, because DNA-computing can “compute” simultaneously. In other words, DNA-computing implements a non-deterministic Turing machine. The same thing happens in membrane computing.

In general, P system consists of three parts: the hierarchical structure of membranes, a multi-set to represent the objects, and the evolutionary rules. The hierarchical structure of membranes defines the computing architecture. It limits the sequence and affiliation of different membranes. The objects’ multi-set represents the developing state during the computing process. The evolutionary

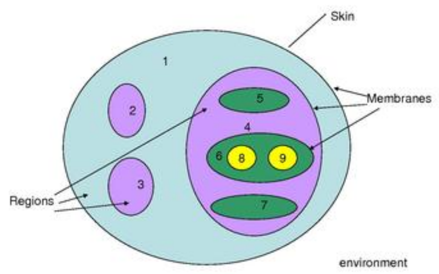

Figure 4.1: Cell membrane structure (from Wikipedia.org).

rules define the method P system uses to develop the results. The hierarchical membrane structure guarantees that the architecture can handle multiple heuristics in one algorithm.

The composite heuristic algorithm based on P system has attracted much attention for two reasons. The first is that the computing model's structure is very suitable for combining multiple algorithms; the other is that the no free lunch (NFL) theorem guarantees its computation capacity.

The famous NFL theorem was proposed and proved by Wolpert and Macready in 1997. It proves "*that if an algorithm performs well on a certain class of problems, then it necessarily pays for that with degraded performance on the set of all remaining problems*" [85]. That is, different algorithms have specific advantages and disadvantages when solving one certain problem. Composite algorithms can synthesize the advantages of different algorithms, which in theory makes them have more searching ability than their individual components considered separately. People designed various mechanisms to avoid heuristic algorithms being trapped into the local optima, such as diversification in TS and mutation in GA. However, being trapped into local optima is still inevitable because of the limitations of the mechanisms. Composite algorithms can benefit from these mechanisms and avoid their limitations as well.

4.1.2 Unit Operations

Before introducing the details of the meta-heuristic algorithm based on P system, the concept "unit operation" needs to be defined.

Unit operations are the manipulations heuristic algorithms use to find new solutions. For most heuristics, they require methods to find new solutions to compare with the current best. The manipulation used in this process is called a unit operation. If we define the concept in a more general way, any manipulation that helps the algorithm find a different solution can be called a unit operation. A formal definition can be given as follows.

Definition 4.1 (Unit Operation)**.** *In a heuristic algorithm, a unit operation is the manipulation(s) that help(s) the algorithm to find a new solution.*

Take local search (LS) as an example. It is a simple and basic heuristic algorithm. At the beginning of the algorithm process, the algorithm randomly generates an initial solution and chooses a method to update it. Then the algorithm keeps using the method to update the current best solution until some stopping criterion is met. The stopping criterion can be any common one, such as a fixed iteration number or a threshold. This method to update the current best solution is the unit operation in this situation. Likewise, in genetic algorithm, the manipulation "mutation" is the unit operation.

Several manipulations are commonly used as a "unit operation" in heuristic algorithms. 2-OPT and 3-OPT are the most widely used. 2-OPT is an abbreviation of 2-optimization. It generates a new solution by exchanging the position of two randomly chosen elements. These two elements can be anything that influences the value of a fitness function. For example, in TSP, when dealing with a path, the element can be a vertex or edge. Likewise, 3-OPT is an abbreviation of 3-optimization. It works by changing the sequence of three elements. If we take the three elements as the vertices constructing a triangle, 3-OPT works by changing the sequence of vertices clockwise or anti-clockwise [86].

4.1.3 Meta-heuristic Based on P System

Considering the concept of a unit operation mentioned previously, the process of P system is not completely parallel. In the process of one region's unit operation, it needs the product of an adjacent region as its initial solution. Understanding this, we can design a complex heuristic algorithm based on P system.

The algorithm based on P system consists of six unit operations, which are six different heuristic algorithms. They are PSO, TS, SA, GA, and two different ACOs. The two ACOs update their pheromone data in different ways. Though ACO2 (called ACO in the previous chapter) produces better results than ACO1 (denoted as AC_Local in the previous chapter) in most circumstances when they are used alone to solve MWLP , we choose to use both of them in the composite algorithm because of the NFL theorem. The algorithm's main structure is shown in Figure 4.2.

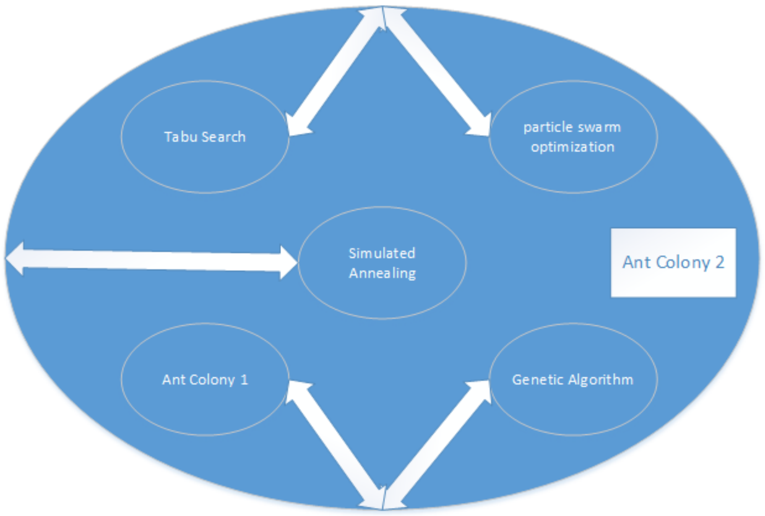

Figure 4.2: Illustration of the algorithm based on the P system algorithm.

As shown in the illustration, the "cell" consists of five inner membranes and an outer membrane. The five inner membranes are independent of each other, whereas they are all connected to the outer membrane. The inner membranes conduct PSO, TS, SA, GA, and ACO1, respectively, and the outer membrane runs ACO2. Previous experiments showed that ACO2 produces the best optimization result among the six heuristic algorithms when they are running alone (same number of iterations) to solve MWLP. Therefore, we consider ACO2 as the most suitable algorithm among the six heuristics and believe it can improve further in fewer iterations. That is why we chose ACO2 as the outer membrane.

When using this meta-heuristic to solve MWLP, the mobile agent's route is stored in vertex arrays. Ten arrays are included in a group. The groups are considered as the "compounds" to transmit between membranes. At the beginning, a group of vertex arrays is generated by using the greedy algorithm. This group is used as the initial solutions of the inner membranes. After the five optimization processes (inner membranes) are performed, each of them outputs a group of vertex arrays, which consists of the 10 best solutions found by itself. The five groups are put together as the initial solutions of the outer membrane. After the outer membrane finishes its optimization process, it outputs a group of the 10 best solutions. This group of solutions is returned to the inner membranes as their initial solutions. At this time, the algorithm comes back to the origin state. We call this an iteration. The algorithm will iterate a predetermined iteration time until the outer membrane outputs the final solution. The procedure's pseudo-code is shown in Algorithm 4.1.

4.1.4 Parallel Technique

It is obvious that the meta-heuristic contains relatively many sub-algorithms that makes its computing scale large. Owing of this, its time consumption is considered as the main drawback of this composite algorithm. However, thanks to the use of membrane computing model, the whole optimization procedure can be designed to run in parallel. The five inner membranes are mutually independent. None of them needs to use the others' outputs.

Based on the analysis above, we designed a parallel operation procedure. Its main program creates five threads corresponding to the five inner-membrane heuristics. They are allocated to different CPU cores to run at the same time. If one thread is done, it sends its result to the main

Algorithm 4.1 *p* System Algorithm

```
1: procedure p-SYSTEM($weighted\_map, priority\_seq, iteration\_time$)
2:     $ini\_seq = GreedyAlg(weighted\_map, priority\_seq)$
3:     $best\_val = ComputeValue(ini\_seq)$
4:     $iteration\_number \leftarrow 0$
5:     while $iteration\_number < iteration\_time$ do
6:         $ac1\_best\_seq = AntColony1Alg(ini\_seq)$
7:         $sa\_best\_seq = SimulatedAnealingAlg(ini\_seq)$
8:         $ts\_best\_seq = TabuSearchAlg(ini\_seq)$
9:         $pso\_best\_seq = ParticlSwarmOptAlg(ini\_seq)$
10:        $ga\_best\_seq = GeneticAlg(ini\_seq)$
11:        $new\_ini\_seq = combination\ of\ all\ the\ five\ seqs\ above$
12:        $ac2\_best\_seq = AntColony2Alg(new\_ini\_seq)$
13:        $temp\_val = ComputeValue(ac2\_best\_seq)$
14:        if $temp\_val < best\_val$ then
15:            $best\_val = temp\_val$
16:        end if
17:        $ini\_seq = ac2\_best\_seq$
18:        $iteration\_number + +$
19:    end while
20:    return $best\_val$
21: end procedure
```

program and kills itself. The main program waits until the five threads are all finished. Then the main program combines the five threads' outputs as the outer-membrane's input. In other words, by running the 6th to the 10th line in the pseudo-code at the same time, some parts of the original algorithm can run in parallel to save time. By using this strategy, the composite algorithm's running time drops impressively without losing computing capability. The experimental results are given later.

4.2 Variable Neighbourhood Descent

VND is a variant of VNS. VNS was first proposed by Mladenović and Hansen in 1997 [87, 88]. Compared with other algorithms, VNS shows its advantages in timeliness and creativeness. It makes VNS very suitable for solving complex problems [89, 90], including NPC problems [91].

Avanthay *et al.* proposed a method based on VNS for graph colouring problems [92]. Ribeiro and Souza solved the same problem by using VND [93]. The proposed VND algorithm performed better than GA in experiments. Algorithms based on VNS and a variant of VNS (RVNS) have been designed to solve VRP [94, 95]. Some algorithms for 0–1 mixed integer programming [96] and minimum spanning tree problems [97] are designed on VNS. In addition to theoretical computing research, VNS has also been applied in real-world applications, including project scheduling problems [98, 99], working arrangement problems [100], and job shop scheduling problems [101]. Another impressive feature of VNS is that it is simple to combine with other algorithms. According to the NFL theorem, the mechanism provides VNS with a wider searching space. The meta-heuristic algorithms combine VNS with GA [101], TS [102], and SA [103] have been proposed and showed good computing capacity and robustness.

VNS is widely used in solving theoretical problems correlated to MWLP. For example, as we mentioned in the introduction, Ban *et al.* used it to solve MLP [16]. In addition, Bjelić *et al.*used VNS to solve a variant of VRP [104]. In this section, we introduce VNS and VND, then go into the details of the algorithm we proposed.

4.2.1 Variable Neighbourhood Search

The principle of VNS is to systematically change the neighbourhood structure during the searching process to expand the searching field to obtain the local optima. Based on this, new local optima can be found by repeatedly performing the same procedure for a different neighbourhood. The global optimum is found by sorting all the local optimal results.

A standard VNS procedure runs as follows.

1. Initialization: Setup the neighbourhood structure sets: N_k ($k = 1, \ldots, k_{max}$) and the stopping

criterion. Initial solution x is also given.

2. Keep performing the steps below until any stopping criterion is satisfied:
   (a) Set $k = 1$;
   (b) Perform the steps below iteratively until $k = k_{max}$
      i. Random Search: Choose x' from x's kth neighbourhood. $x' \in N_k(x)$.
      ii. Local Search: Take x' as the initial solution, apply local search method to find the local optima x_l^*.
      iii. Update: If x_l^* is better than the current best solution, update $x = x_l^*$, keep performing local search in neighbourhood N_l. Otherwise, set $k = k + 1$.

Figure 4.3 illustrates the procedure of a basic VNS. The algorithm keeps moving its searching space to a new neighbourhood to expand its possibility of finding the global optima.

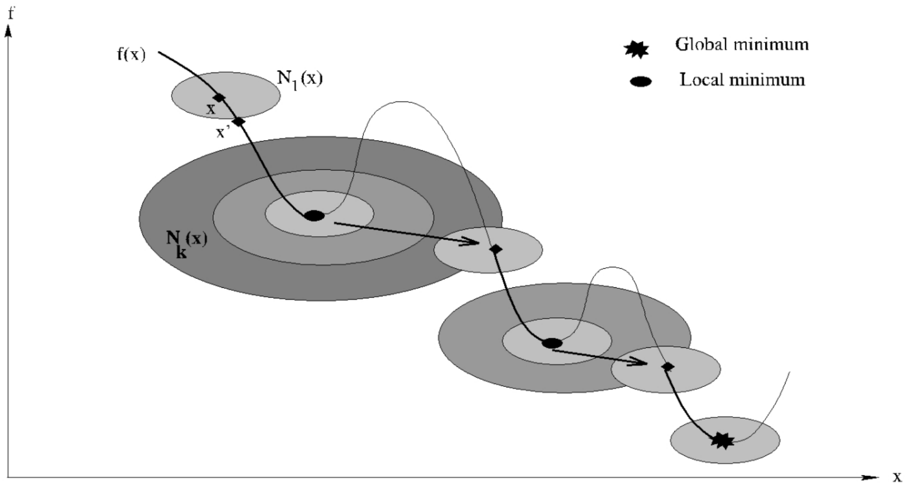

Figure 4.3: Basic VNS from [105].

In real-world applications, the searching procedure could be trapped in the procedure “Local Search” (2.b.ii). To prevent this shortcoming, many variants of VNS have been designed. VND is one of these.

4.2.2 Variable Neighbourhood Descent

VND was also proposed by Mladenović and Hansen. They explained the model in 2000 [106]. In general, it skips the procedure "Random Search" (2.b.i) and performs the procedure "Update" (2.b.iii) in a deterministic way. In other words, the procedure "Local Search" is replaced by a heuristic strategy. The procedure (2.b) is changed as follows.

1. Local Search: Perform heuristic by using x' as its initial solution. The solution is denoted by x_l^*.
2. Update: If x_l^* is better than the current best solution, update $x = x_l^*$. Set $k = k + 1$.

The pseudo-code of VND is given in Algorithm 4.2 "Basic VND".

Algorithm 4.2 Basic VND

```
1: procedure Basic VND($x, k_{max}$)
2:     while 1 do
3:         $k \leftarrow 1$
4:         while $k < k_{max}$ do
5:             Find the best neighbour in $x$'s neighbours
6:             Change $x$ and its neighbours to the better ones
7:         end while
8:         if no improvement is made
9:     end while
10: end procedure
```

Compared with VNS, VND has two main improvements. It inherits the advantages of multiple neighbourhood structures from VNS, which makes it have more chance of obtaining global optima [105]. On the other hand, it eliminates the procedure "Random Search", which saves a substantial amount of time.

4.2.3 Meta-heuristic Based on VND

As for MWLP, it is obvious that the final result is affected by two key features: the weight of vertices and the distances between vertices. The vertices with higher weights and the edges with shorter distances are prone to be chosen earlier. Knowing this, we specifically designed an algorithm based on VND for MWLP. This algorithm divides the whole searching space into different neighbourhoods by using different x' as initial solutions of procedure (2.b). The initial solutions are generated as follows.

We divide the vertex weight interval $(0, 1]$ into multiple intervals by using a prefixed value *interval_length*. The value determines the section length. All the vertices whose weights are in the same interval are grouped together. In this way, all the vertices are divided into several groups. The vertices in one group are considered together, and they will produce a visiting array of this group. For each group, the vertices can be seen as small weighted complete graphs. The two vertices with the shortest edges are chosen as the first two vertices to be visited. Among these two vertices, the one with higher weight value is visited first. The second vertex chooses the vertex nearest to it among the vertices left in this group as the next visiting vertex. By using this method, the group's visiting array will be generated. Namely, the previously chosen vertex will choose the vertex nearest to it as the next visiting vertex among the unvisited vertices.

After every group's visiting arrays are all generated, the groups will be sorted in descending order by their weights. The groups with higher weights are visited earlier. The vertices within one group are visited in the generated order. Then we obtain a visiting order that consists of all the vertices. It is used as the initial solution in procedure "Local Search" and, at the same time, it limits the concept of "neighbourhood". For the procedure "Update", any optimization algorithm can be used to enforce the searching procedure.

New neighbourhoods are generated by moving the group positions. They are moved from left to right by a short pre-determined length *delta* every time. The whole process ends when the interval number shrinks to one. Namely, the interval returns to the original, which is $(0, 1]$. The group moving process is illustrated in Figure 4.4 and the pseudo-code is listed in Algorithm 4.3. In the pseudo-code, a value *initial_point* is added in case some other requirement is needed. The value is set to zero in common cases.

4.2.4 Parallel Technique

The value *delta* is very important for this algorithm based on VND. It should not be very small, because every group position change is supposed to lead to a visiting array change. If it is too small, two neighbourhoods could be the same, making the change meaningless. It should also not be too large. Otherwise, the structure of the neighbourhoods will be very loose. The algorithm may not be able to search enough neighbourhoods to find a better result. Our experiment result shows that a suitable *delta* value is 1/vertex_number. Take the instance with 100 vertices as an example, *delta* will be set to 0.01. That means the algorithm will have 100 neighbourhoods to search. If we perform these search processes in sequence, the procedure "Update" will be executed 100 times.

Fortunately, the neighbourhood searching processes do not have to be performed in sequence. Similarly to the inner membranes in P system, the neighbourhoods are also mutually independent. We can allocate them into different threads to let them run at the same time to cut the running time. When the computer has adequate CPU resources, the efficiency improvement is impressive. The experimental results are given in the following section.

Algorithm 4.3 VND Algorithm

```
procedure VND($initial\_point, interval\_length, delta$)
    $i \leftarrow 0$, $best\_val \leftarrow inf$
    while $initial\_point + i * delta < 1$ do
        $count \leftarrow 0$
        while $initial\_point + i * delta + count * interval\_length < 1$ do
            if $initial\_point + i * delta + (count + 1) * interval\_length < 1$ then
                $left\_margin = initial\_point + i * delta + count * interval\_length$
                $right\_margin = initial\_point + i * delta + (count + 1) * interval\_length$
                $sort\_interval(count) = [left\_margin, right\_margin)$
                $vertex\_neighbor(count) \leftarrow all\ the\ vertices\ in\ sort\_interval(count)$
            else
                $left\_margin = initial\_point + i * delta + count * interval\_length$
                $sort\_interval(count) = [left\_margin, 1]$
                $vertex\_neighbor(count) \leftarrow all\ the\ vertices\ in\ sort\_interval(count)$
            end if
            $count + +$
        end while
        for $i = 0 : count$ do
            $greedy\_seq(i) = GreedyAlgBasedOnDistance(vertex\_neighbor(i))$
        end for
        $initial\_seq = [greedy\_seq(0), greedy\_seq(1), \ldots, greedy\_seq(count)]$
        $temp\_val = HeuristicAlg(initial\_seq)$
        if $temp\_val < best\_val$ then
            $best\_val = temp\_val$
        end if
        $i + +$
    end while
    return $best\_val$
end procedure
```

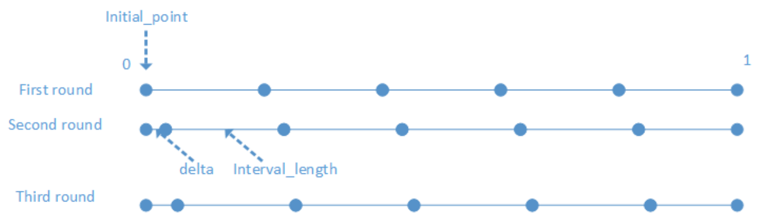

Figure 4.4: Illustration of the algorithm based on VND.

4.3 Experimental Results

To test the efficiency and effectiveness of the two proposed algorithms, we designed several experiments. The experimental settings and environment remain the same.

In the previous chapter and [107], we showed that classic heuristics could be used to solve MWLP. ACO with a global pheromone updating mechanism (ACO2 in P system) is ranked first among the tested heuristics. DP is the only algorithm whose efficiency is acceptable and that can find the exact solution when the problem scale is relatively large (20 vertices). It was tested in [107] as a comparison when the number of vertices was 20. In this chapter, problem instances with 20, 30, 50, and 100 vertices are tested. They are considered as examples of small-scale (instances with 20 vertices), medium-scale (instances with 30 and 50 vertices), and large-scale problems, respectively. For the situations with 20, 30, and 50 vertices, we used the same graphs as in the previous chapter and [107]. For the graph with 100 vertices, because of some Matlab® inner random number generation mechanism, its vertex distribution structure is very similar to the graph with 20 vertices. Thus, we regenerated the graph with 100 vertices to guarantee its structure is different from the other instances. This should reduce the influence of the graph structure on the effectiveness of the algorithm. As we did in the previous chapter, all the graphs were also generated by randomly positioned vertices in a 200×200 area, and their weights are allocated randomly. All the algorithms under different instances were tested 100 times to make sure the experimental results could show the average performances of algorithms. The results of ACO and DP (only for the instance with 20 vertices) were used for comparisons.

As for the parameters, for the algorithm based on P system, the number of iterations within different membranes was all set to 50. The algorithm circulated five times between the outer membrane and the inner membranes. For the algorithm based on VND, ACO1 was chosen as the enforcing heuristic. The number of iterations was set to 50. As stated in the previous section, *delta* was set to 1/vertex_number. Intuitively, ACO2 should be chosen for the enforcement because it is considered as the most powerful heuristic when solving MWLP. However, the experiments produced different but interesting results. More details are given in the analysis section.

The experimental results are presented in Figures 4.5–4.8. All the figures contain the results for five algorithms: ACO, P system sequential version, P system parallel version, VND based on ACO1 sequential version, and VND based on ACO1 parallel version. In the figures, they are represented as ACO, p_system_seq, p_system_par, VND_ACO1_seq, and VND_ACO1_par, respectively. In general, the sequential and parallel versions of an algorithm have very similar effectiveness. The efficiency is their main difference: the parallel version drastically improved the algorithm efficiency, especially for the algorithm based on VND.

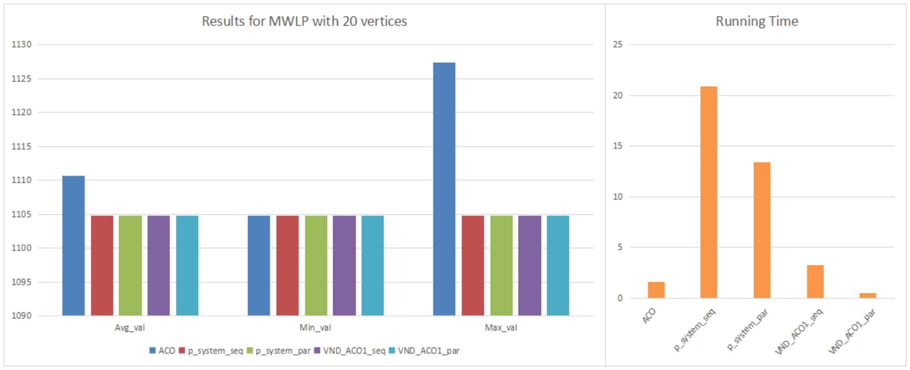

Figure 4.5: Experimental results for the instance with 20 vertices.

DP was executed in the experiments for the instance with 20 vertices. Thus, we know the global optima here. Among the five tested heuristic algorithms, the proposed algorithms found the global optima for every test. ACO obtained 22 hits within 100 tests. Obviously, VND_ACO_par is the best algorithm in this situation. It not only had the shortest running time, but also guaranteed to find the global optima every time.

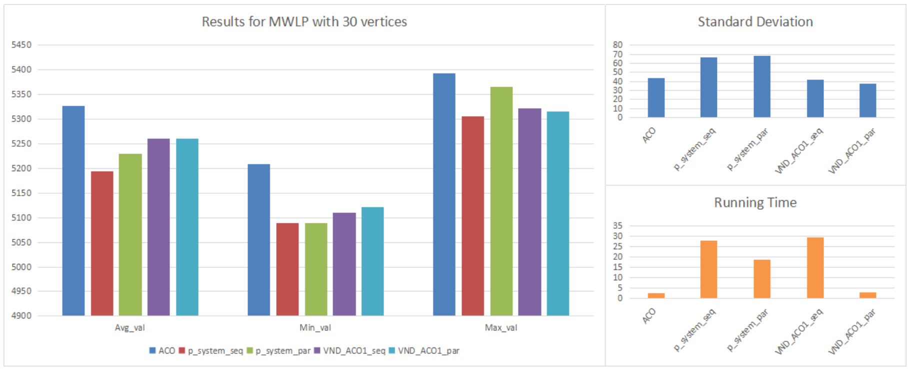

Figure 4.6: Experimental results for the instance with 30 vertices.

As for the other small-scale problem instance, the result in the problem with 30 vertices was not surprising. VND_ACO_par still had the shortest running time. Compared with ACO, the four versions of the proposed algorithms all improved the result by around 3%. P_system_seq obtained the best result among the five algorithms; however, its running time was the longest.

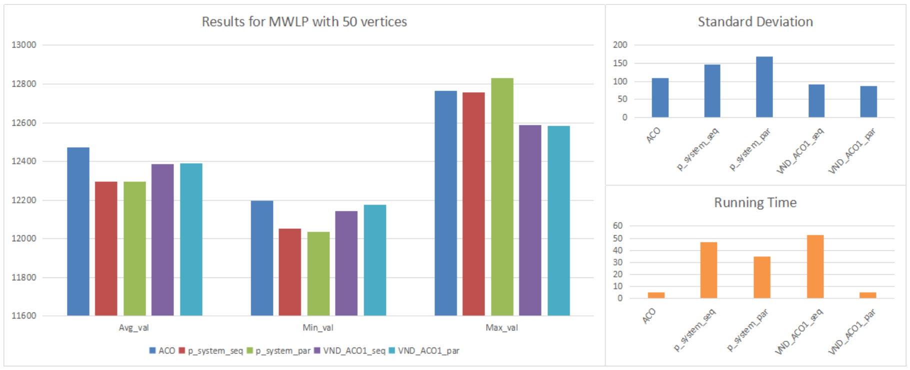

Figure 4.7: Experimental results for the instance with 50 vertices.

The instance with 50 vertices is considered as a medium-scale problem. When thinking about the efficiency, VND_ACO_par again ranks first. Compared with ACO, the algorithms based on P system and VND slightly improved the result by around 1.5%. However, the improvement is not as significant as when the problem scales are smaller or larger.

For the experiment with 100 vertices, again VND_ACO_par consumed the shortest time. For

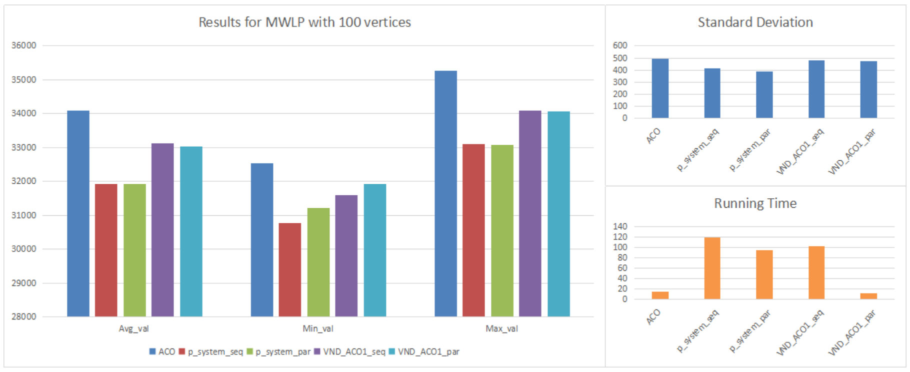

Figure 4.8: Experimental results for the instance with 100 vertices.

the algorithm effectiveness, the algorithms based on P system improved the result by 6%. The algorithms based on VND improved the result by around 3%. That is, when MWLP's problem scale becomes large, the algorithm based on P system should be chosen if a better result is required. However, if a relatively good result is needed in a short time, VND_ACO_par is the best choice.

4.4 Theoretical Analysis

The experimental results are very interesting. In this section, we provide an analysis of different aspects from the theoretical viewpoint. The contents will cover the efficiency improvement by using parallel technologies, the performance differences of VND algorithms based on AC1 and AC2, and the reason for the good performance with regards to effectiveness of P system and VND.

4.4.1 Efficiency Improvement by Parallel Mechanism

By manipulating the parallel technique, the running times of both proposed algorithms are reduced. Tables 4.1 and 4.2 illustrate the effect.

For the algorithm based on P system, its parallel version had huge efficiency improvements in all different problem scale instances. However, as the problem scale increased, the efficiency

	Time_Seq	Time_Par	Time reduction
20 vertices	20.8789 s	13.3894 s	35.87%
30 vertices	28.1119 s	18.8152 s	33.07%
50 vertices	46.9584 s	34.7607 s	25.98%
100 vertices	119.7665 s	95.4062 s	20.34%

Table 4.1: Running time comparison of the algorithm based on the sequential and parallel versions of P system.

improvement decreased. This could be explained from three aspects. One is that this algorithm is not evenly separated. The five inner membrane threads' running times are very different. When the faster threads finish their job, the slower threads are still running. The second is that the threads are not running at the same time. The outer membrane thread must wait until all inner membrane threads are finished. The last is the algorithm's scale. When dealing with large problem instance, the computer's internal memory is not sufficient to satisfy the requirements of the algorithm. The external memory will be used to extend the memory capacity. However, because the speed of external memory is much slower than internal memory, the running time of the algorithm will be greatly increased. As the problem size increases, more and more external memory will be used, which will make the algorithm run slower. This explains why the efficiency of the parallel version of the algorithm improved less as the problem size increased.

	Time_Seq	Time_Par	Time reduction
20 vertices	3.2890 s	0.4563 s	86.13%
30 vertices	29.7497 s	2.8851 s	90.30%
50 vertices	52.4615 s	5.0067 s	90.46%
100 vertices	102.9535 s	11.0290 s	89.29%

Table 4.2: Running time comparison of the algorithm based on the sequential and parallel versions of VND.

Table 4.2 shows the running times of the algorithms based on VND. The efficiency improvement of the parallel version is very impressive. Compared with the sequential version, the algorithm based on the parallel version of VND only consumed about one-tenth of the time. Surprisingly, attaining this significant improvement in efficiency, the algorithm could still output as good a result as its sequential version. This is because its structure is very different from the algorithm based on P system. The running time of its neighbourhoods are almost the same and do not have to wait for each other to continue. What is more, the search spaces are relatively small. The algorithm based on VND perfectly avoids the drawbacks of its counterpart. Its features guarantee that all the search processes start as soon as a CPU core is able to deal with a new task. On the other hand, with a small algorithm scale, the external memory does not have to be involved in the searching process. These advantages speed up the manipulation process of the algorithm.

4.4.2 Effectiveness Differences of VND Based on AC1 and AC2

Both algorithms based on P system and VND performed well under all vertex number instances. For the algorithm based on P system, we have previously used NFL theorem to explain its search capacity. Everything happened as expected. However, something interesting happened when dealing with the algorithm based on VND.

Before discussing the effectiveness of the algorithm, we would like to discuss some aspects of ACO. Different ACOs have different mechanisms to update the pheromone table. In [107] and Chapter 3, we tested two pheromone updating mechanisms. One updates its pheromone locally (denoted as AC_Local in the last chapter), which means it updates its pheromone table as soon as it finds its next step. The other updates its pheromone globally, which means it updates its pheromone table when it finishes an iteration and updates the table by using the step's best solution. We call the ACOs using these two mechanisms AC1 and AC2, respectively. AC2 is also the ACO algorithm we used in this chapter. In the previous chapter, our experimental result showed that AC2 produced better results than AC1 in medium-scale problems. However, the results were limited to medium-scale problems. Moreover, we have only shown that the output of AC1 was worse than AC2. Considering the fact that AC2 performed as one of the best algorithms, we still do not know how bad AC1's performance was. Thus, we would like to present a more comprehensive comparison of the search capacities AC1 and AC2. In addition, the iterated local search (ILS) method will be added as a comparison.

The structure of ILS is relatively simple. Simply speaking, it iteratively performs the local search process (2-OPT for example) for the current best solution. The final output is produced after some stopping criterion is met. The results of the experiments given in the following.

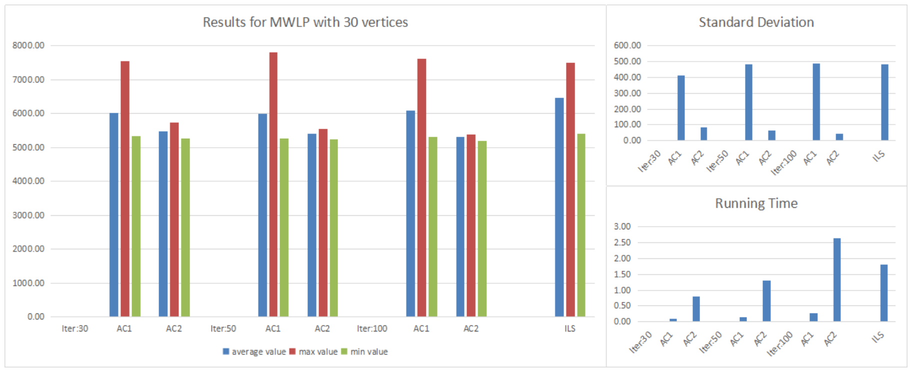

Figure 4.9: Experimental results for different ACOs and ILS solving MWLP with 30 vertices.

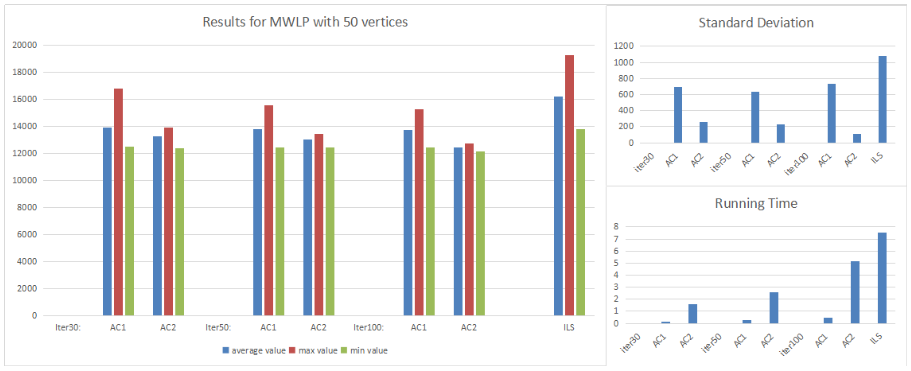

Figure 4.10: Experimental results for different ACOs and ILS solving MWLP with 50 vertices.

Figures 4.9–4.11 show the experimental results using AC1 and AC2 to solve different MWLPs with 30, 50, and 100 vertices, along with the results of ILS. The stopping criterion for ILS was

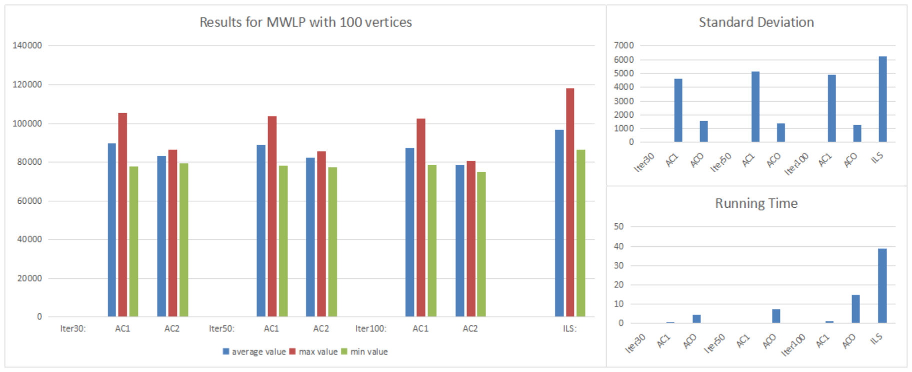

Figure 4.11: Experimental results for different ACOs and ILS solving MWLP with 100 vertices.

set as the best solution remains unchanged in 10,000 iterations. Its unit operation was 2-OPT. The results show that ILS is too simple to solve MWLP when the problem scale is relatively large. In terms of AC1 and AC2, from the three dimensions we used to evaluate the effectiveness of a method, they performed better than ILS. However, they have minor differences. AC1's standard deviation values and its worst results were worse than AC2's. AC1's performances were not as stable as AC2's. However, as we showed in the previous chapter, AC2 tends to be trapped in local optima. Its high stability strengthens this tendency. Though AC1 cannot guarantee finding a good result in every test, it provides the possibility to search the space out of local optima.

As we have explained, AC1 was used as the enforcing searching process after the neighbourhoods were generated. Intuitively, AC2 should be the better choice. After all, it demonstrated better search capacity than AC1 when they were used alone to solve MWLP. Nevertheless, experiments are still needed to test which ACO is more suitable for solving MWLP.

Figure 4.12 shows the comparison results. We did not show the results for 20 vertices here because both algorithms can find the best result in every test. For the instance with 30 vertices, the algorithm using AC2 produced better results. However, when the number of vertices increased to 50, the algorithms using the results of AC1 and AC2 are almost the same. When the number

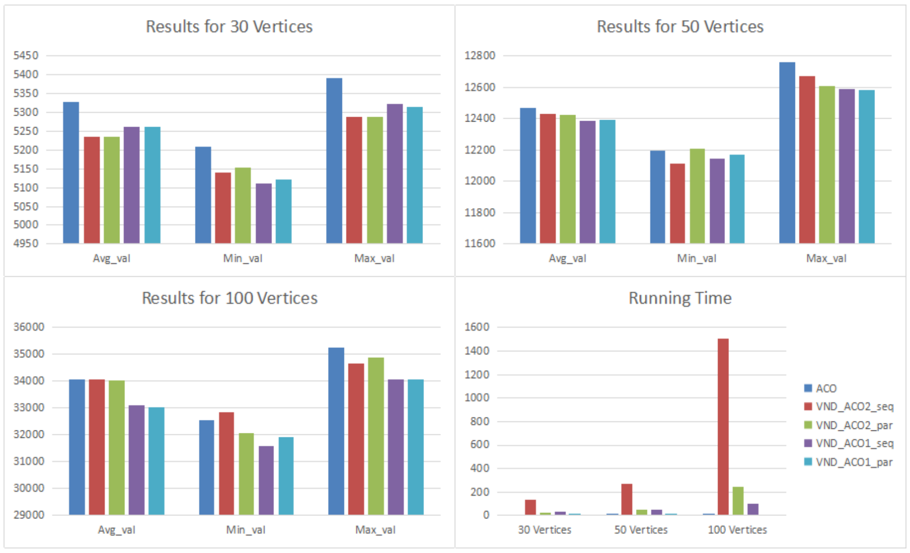

Figure 4.12: Experimental results for the VND algorithms using AC1 and AC2.

of vertices increased to 100, the algorithm using AC1 had better results. In other words, as the problem scale increases, the algorithm using AC1 achieves better results.

This should be caused by the interference of VND structure. When the problem sizes were small, the search space was also small. Both AC1 and AC2 alone have sufficient search capacity to find the global optima. Thus, in this situation, the results mainly show the search capacity of AC1 and AC2, but not the VND structure. As the problem scale increases, neither AC1 or AC2 can obtain the global optima. VND was used to help divide the search space into small parts. This mechanism shrinks the size of search space and should limit AC1 and AC2 within this small space. Thus, both AC1 and AC2 can fully use their search capacity. However, the pheromone updating mechanism of AC2 helps it to jump out of the local optima when it is used alone. Ironically, in this situation, it goes in the opposite direction. This mechanism mostly offsets the advantage of the use of the VND structure. For AC1, it keeps searching in its small space restricted by the VND structure, which helps it to perform the deep searching to obtain a better result. When used in the algorithm based on VND, the pheromone updating mechanism of AC1 is more effective, especially for the large-scale problem instances.

4.4.3 A Hypothesis

Compared with the classic heuristics tested in the previous chapter, both meta-heuristics showed better searching capacities. With the increase of the scale of the problems, this advantage became more convincing. At the beginning of this chapter, it was noted that the NFL theorem guarantees the search capacity of combining different heuristics. Here we introduce a hypothesis about the search capacity meta-heuristics demonstrated when solving MWLP from a different aspect. That is, different from normal NPC problems, MWLP has two parameters. This feature leads to the probability that MWLP has more uncertainties, so the algorithms designed for MWLP should be able to cover different aspects referring to their search capacities. According to the NFL theorem, different heuristics provide different search capacities. No single heuristic can satisfy the whole search capacity requirement of MWLP. In other words, a complex meta-heuristic should be more suitable when solving MWLP. For the P system and VND algorithms, each combines different heuristics. This mechanism guarantees they have wider search capacities that exactly meet the requirements of MWLP.

The value of the objective function of MLWP is influenced by two variables. Both the distances between vertices and the weights of vertices play important roles. This is the major reason why MWLP looks more complex than ordinary NP-hard problems. The NP-hard problems we usually deal with, such as TSP or MLP, are problems whose final solutions are influenced by one variable. Take TSP as an example: the objective of TSP is to find a Hamiltonian tour that minimizes the distance the salesman travels. This distance is merely influenced by the paths the salesman travels. However, in MWLP, the salesman must consider both the paths he travels and the vertices he passes. If a vertex is skipped by the salesman, the influence on the final result is different for TSP and MWLP. For TSP, the salesman only needs to deduce the paths before and after the vertex he skips, and then adds the distance between the vertices before and after the vertex he skips. However, for MWLP, the salesman must consider all the changes in latencies of weighted vertices.

For the heuristics, different heuristics have different advantages. According to the NFL theorem, the advantages remain unknown before we test them. However, even though we have tested them, we do not know what kind of core mechanism made the heuristic show its advantages in some certain problems. For example, the experimental results show that ACO and GA are the best among the classic heuristics. The reason why GA and ACO performed well remains unknown. This is also the main idea of the NFL theorem.

However, if we combine the features of MWLP and the NFL theorem, the reason why meta-heuristics show better searching capacities is clear. MWLP has twice the number of variables, which leads to more uncertainties. This feature requires the algorithms to be more diverse in searching capacities, which is exactly what a meta-heuristic that contains multiple heuristics or different searching skills can provide.

4.5 Conclusions

In this chapter, we have proposed two new meta-heuristics to deal with MWLP. The parallel techniques were used to improve the efficiency of algorithms. The effectiveness of algorithms were tested under small, medium, and large problem instances. Experimental results proved that both meta-heuristics could produce better results than single classic heuristic. As for the small-scale problem, the algorithm based on P system is more suitable. When the problem scale increases, considering the tradeoff of efficiency, the algorithm based on VND is the best choice.

5 Real-World Application

In previous chapters, we introduced different methods to solve MWLP, including exact algorithms and several heuristics. In addition, the methods are tested under different circumstances. However, we have still not explained the importance of MWLP from the application aspect. In this chapter, we focus on the real-world applications of MWLP.

Every NP-complete problem has its own features when studying it in theory. In addition, its features make it more suitable to modelling some certain kinds of real-world applications. MWLP was originally derived from treasure hunts. In modern times, the job of Indiana Jones is not as popular as it was several hundred years ago. We could still find applications in our daily lives. During our research, we found MWLP can handle more extensive problems than MLP because of the use of variable vertex weight. In this chapter, we introduce two real-world instances as examples that are modelled and solved by MWLP. They are the applications of MWLP in emergency relief and perishable commodity transportation.

Note that in real-world applications, the distances between vertices may not fit triangle inequality, nor the graph be complete at all. In the former case, the problem can be handled by maintaining a table which stores the shortest distance values between all the vertices, and the routes to achieve the distances. When solving MWLP, the distances between different vertices should be changed by first hand according to the table. After getting the result by solving the map-changed MWLP, check the table to output the final route. As for the case where the graph is not a complete graph, all the unconnected edges could be set to infinite during the coding process. If the "inf" value is not optional, the distance can be simply set to be a relatively large value to guarantee the unconnected edges will not be chosen by the algorithm.

5.1 Emergency Relief

5.1.1 Background and Literature

Natural disasters have accompanied human beings since the beginning of human history. Ancient floods were recorded in the literature from different cultures all over the world, from Noah's Ark to the story of King Yu, who tamed the flood in Chinese mythology. In the modern age, the influence of large natural disasters on human beings has not ceased. In 1999, a flood in Venezuela killed 30,000 people[108]. In 2004, the earthquake and tsunami in the Indian Ocean caused the death of almost 300,000 people[109]. Figure 5.1 shows pictures of a coastal area before and after it was ravaged by the tsunami. The comparison is shocking. In 2008, 3 months before the Olympic Games held in China kicked off, the earthquake in Szechwan province led to the death of more than 150,000 people[110]. The United Nations declared the 2010 Haiti earthquake, which killed more than 220,000 people, as the most severe disaster it had ever encountered only a few days after the earthquake happened[111].

Such disasters are destructive. They injure people, destroy buildings, and cause shortages of food and drinking water. Usually, soon after a disaster happens, relief goods including shelter, food, and medicine will be provided to the victims by the local government or United Nations. Other international federations including the Red Cross, Red Crescent, and other non-governmental organizations will also participate in the rescue to provide and allocate more goods and materials later. With the development of modern technology, the emergency relief work has become more efficient and effective. More food and specific targeted medicines can be delivered to victims quickly. Nevertheless, considering the massive number of people influenced by severe disasters, the relief goods are relatively limited. On the other hand, the possibility of the post-disaster

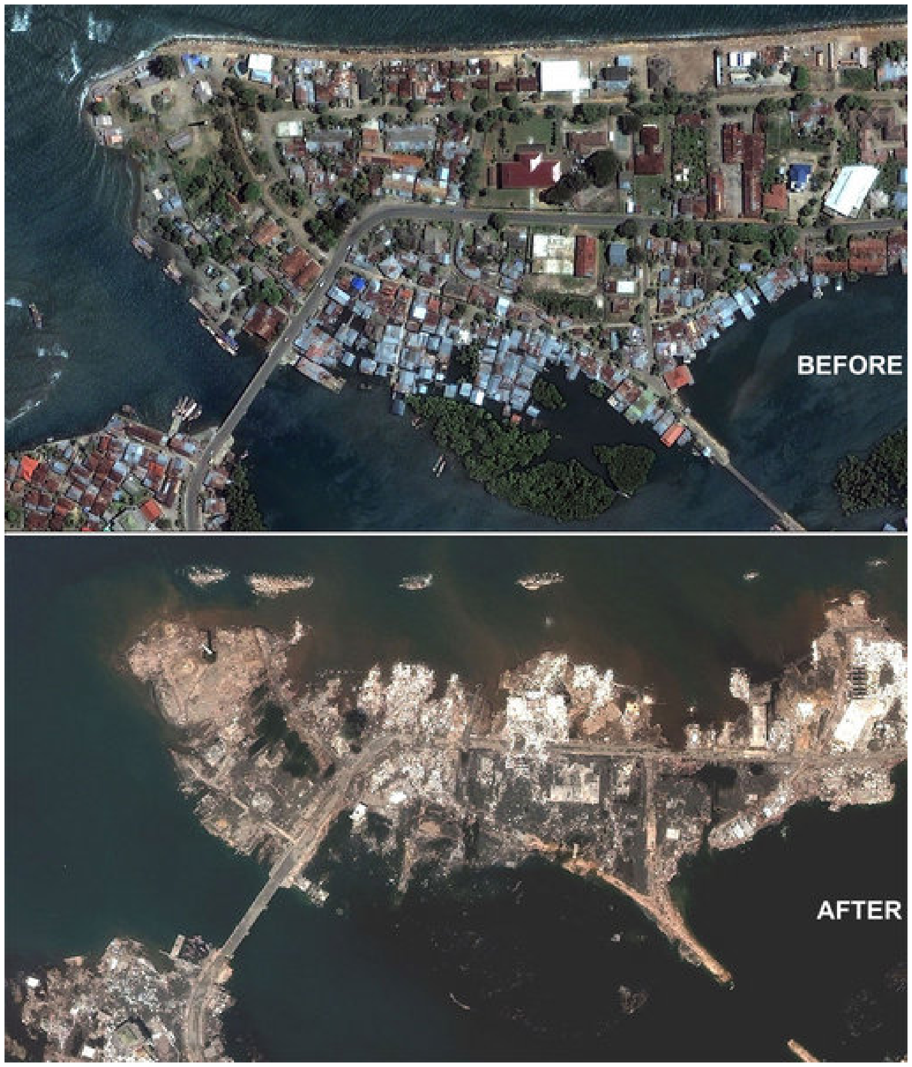

Figure 5.1: Influence of a tsunami on a coastal area [112].

Character	Number of papers
Min Cost	15
Min Unsatisfied Demand	15
Min latest arrival time	4
Min total response time	11
Max travel reliability	2
Stochastic supply	3
Stochastic demand	6
Multi-commodity	13
Multiple depot	7
Single depot	15
No depot	7
Heterogeneous vehicles	18
Stochastic travel time	5
Data from real disasters	19

Table 5.1: Summary of characteristics in disaster relief distribution models [114]

outbreak of communicable diseases is extremely time sensitive [113]. These concerns require the transportation process to be more efficient and targeted. A good route design strategy for the relief vehicles would be helpful.

In 2005, United Nations established the Logistics Cluster to help improve the effectiveness and efficiency of logistic response in humanitarian emergency missions, which is considered as a sign that United Nations had realized the importance of logistics in aid operations and started to try to find a solution [114]. Meanwhile, researchers have also presented many studies on this topic. In [114], the authors showed a summary of characteristics in disaster relief distribution models based on 29 papers published from 1987 to 2011. Table 5.1 shows the characters and the numbers of papers studied on it.

As shown in Table 5.1, the study of goods distribution routing for emergency disaster relief usually models the problem from four aspects, which are the objective function, goods, routing, and test data. The studies of goods and test data are out of the scope of this thesis. We give a brief introduction to the objective functions and routing design, then go to the application instance.

The objective function represents the target of the optimization process. Most of the papers had multiple objectives. The minimum cost and minimum unsatisfied demand were the two objectives used most frequently. Of these two objectives, more papers chose the minimum cost. They tried to minimize the total cost in the goods distribution process, which is fairly straightforward in most situations. However, considering the situation of emergency relief, saving money should not be chosen as the top priority. In addition to the minimum cost, half of the papers paid attention to minimizing the maximum unsatisfied demand of all victims. Compared with minimizing the cost, this factor is more suitable. It is considered from the side of the victims, and it tries to balance the losses from different areas. After all, in disaster relief, it is widely accepted that equity should be the first priority [115]. In addition, by considering the victims, the problem turns into a custom-oriented problem. This leads to the idea that MWLP can be used to help model the problem in this situation.

For the routing design, the papers face different kinds of problems. Usually, the goods are distributed from multiple depots. However, the infrastructure construction quality of some disaster-affected areas can be low. In this situation, the vehicles must depart from fewer or even single depots. During the distribution, the vehicles differ in terms of speed, capacity, and personnel allocation. In addition, the travel time can be uncertain because of the disaster. All these considerations can make the problem extremely complex. In this thesis, we focus on problems with a single depot and one vehicle.

5.1.2 Application Instance

As discussed above, different studies on disaster relief are based on selected situations. No unified standards can be found. Thus, we designed a situation and modelled it using MWLP. This hypothetical situation is largely based on an earthquake that happened in Qinghai province of China in 2010.

Suppose an earthquake affects an area that contains several villages. The houses in this area are made of mud-brick. Even though the houses collapsed, they did not cause severe injuries to or the death of residents. Therefore, many people survived the earthquake. However, if the goods cannot be supplied in time, the victims may die because of the shortages in shelter and food. As described above, to simplify the problem, we limit the emergency relief crew to consist of one vehicle and one depot. In addition, we ignore the time the crew spent allocating the goods in villages.

Now we need to introduce the concept of casualty rate in unit time (CRUT). Consider the situations that victims die because of the shortages in food and shelter. The situations vary in different areas because of different levels of development and degrees of disaster. In a short period, the number of casualties can be hypothesized as increasing linearly over time. CRUT describes the urgency level of an area that needs help. Usually, the unit of time is minutes.

"Minimizing the number of casualties" is chosen as the objective in this model. We believe that saving as many people as possible is the first principle in the disaster relief process. For comparison, a strategy called the nearest neighbour (NN) strategy introduced by Lee [116] is chosen. It is an approach commonly used in dispatching ambulances and proved effective in the literature. The NN strategy is a greedy scheme. The ambulance only reacts to its nearest call. In our experiment, the strategy is implemented as the rescue vehicle goes to the nearest disaster-affected village first.

Coord:	village1	village2	village3	village4	village5	village6	village7	village8
X axis:	110	57	151	114	11	156	26	94
Y axis:	183	151	76	15	106	187	114	2

Table 5.2: Coordinates of the depot and villages.

village1	village2	village3	village4	village5	village6	village7	village8
1	1/4	1/2	1/3	1	1/3	1/2	1/3

Table 5.3: Different villages' CRUT values.

The following is the instance we designed and tested. It shows how to apply the MWLP model in disaster relief. Suppose a disaster has an influence on eight villages in a 200 km $\times$ 200 km area. The coordinates of the villages are listed in Table 5.2. Now, one vehicle is moving out from the depot to transport the relief goods to the eight villages. The depot's coordinates are $(166, 117)$ and the vehicle's speed is 60 km/h. To save space, the coordinates shown in the tables are rounded.

Different villages' CRUT values are listed in Table 5.3. Those values are not the real increased number of the deceased people in unit time. They have been processed by performing data normalization. Take the CRUT values of the second and the third village shown in the table as an example: they are 1/4 and 1/2, respectively. In our model, this does not mean the disaster causes one villager to die every 4 minutes and 2 minutes, respectively. These numbers represent that in the same period, the third village will have twice the number of casualties as the second village. CRUT represents the relative level of damage in different villages. This variation of CRUT values has no influence on the performance of equity in our model, which is the first principle in emergency relief as mentioned above. In our experiment, we assume that the relief crew carries enough relief goods. Thus, as soon as the crew arrives at the village, all the living people in that village will survive.

Figures 5.2 and 5.3 show the routes produced by using MWLP model and NN strategy, respectively. The red circle represents the depot and the stars represent the villages. Different villages can be distinguished from each other by their coordinates. And the arrows illustrate the vehicle's moving direction.

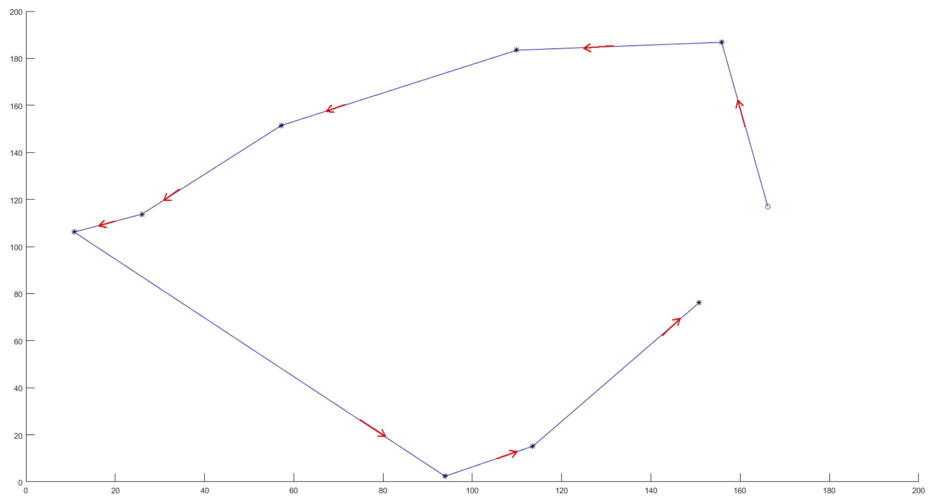

Figure 5.2: Vehicle route calculated by the MWLP model.

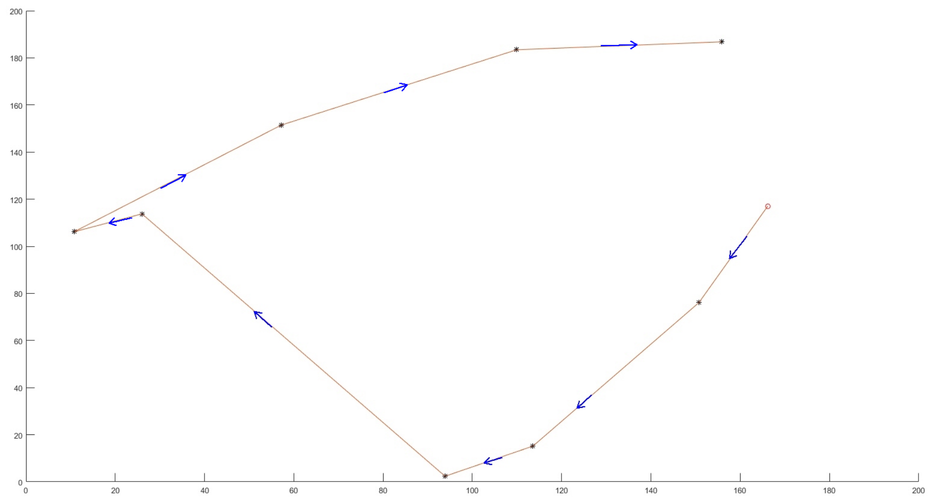

Figure 5.3: Vehicle route calculated by the NN strategy.

As shown Figures 5.2 and 5.3, the routes produced by two schemes are different. The route given by MWLP is: $depot \rightarrow village6 \rightarrow village1 \rightarrow village2 \rightarrow village7 \rightarrow village5 \rightarrow village8 \rightarrow village4 \rightarrow village3$. The route given by using the NN strategy is: $depot \rightarrow village3 \rightarrow village4 \rightarrow village8 \rightarrow village7 \rightarrow village5 \rightarrow village2 \rightarrow village1 \rightarrow village6$. Though both schemes have some shared paths, the directions of the paths are different.

If the relief crew takes the route produced by MWLP, there will be 1037 casualties after they visit every village. This route takes the crew about 472 minutes to arrive at the last disaster-affected village. If they take the other route, there will be 1180 casualties. The time consumption will be 459 minutes. Though they may save 13 minutes to finish visiting all the villages, the casualty rate increases by 13.8%.

This instance is generated randomly and very simple. The real-world situation can be much more complicated. However, it still shows the possibility of using MLWP to solve the disaster relief problem and has proved that MWLP is helpful. More features can be added to help MWLP solve more complex problems.

5.2 Perishable Commodity Transportation

We explain a perishable goods transportation problem in this section. It is similar to the problem introduced in the first chapter. We expand Bob's problem to a situation that can be used more widely. The details are described in the following.

Suppose a supermarket chain has one depot and six stores in one city. The depot and stores are located in a 20 km $\times$ 20 km area. Their coordinates are listed in Table 5.4. Every 2 hours the depot despatches a truck to transport commodities to different stores to fill each store's shortages. The truck's speed is set to be 60 km/h. Usually, different stores require different commodities. Some of the commodities are perishable. For example, some stores may require fruit, others may want fish. Suppose the corporation has no extra money to invest in its own cold-chain transportation network. They must face the problem that the fruit may spoil and the fish may die during transportation. In this situation, they would like to minimize their loss during the transportation by designing a better route for the truck.

Coord:	depot	store1	store2	store3	store4	store5	store6
X axis:	0.7	18.7	15.1	7.8	3.4	0.6	0.9
Y axis:	17	13.6	14.8	13.1	14.1	5.5	1.9

Table 5.4: Coordinates of the depot and stores.

Different kinds of fruit have different prices and different spoilage times. Different species of fish may die at different rates. Thus, the losses of the various perishable commodities during the same period are different, and they vary by time. Suppose we can find a method to model the loss over time as a linear function. Similar to the definition of CRUT given above, we use loss rate in unit time (LRUT) to represent the slope of the linear function. To simplify the problem, we limit every store to require only one kind of commodity. The LRUT values of the commodities required by the six stores are listed in Table 5.5.

store1	store2	store3	store4	store5	store6
1/3	1/2	1/4	1	1/2	1/2

Table 5.5: The LRUT values of the commodities required by different stores.

For the experiment, we set the LRUT values of the stores as the weights of vertices in the MWLP model. Obviously, the value of MWLP's objective function is the same as the supermarket chain's total loss during the transportation. The NN strategy is also implemented for comparison. The routes produced by the different schemes are shown in Figures 5.4 and 5.5, respectively. The route given by MWLP is: $depot \rightarrow store4 \rightarrow store5 \rightarrow store6 \rightarrow store3 \rightarrow store2 \rightarrow store1$.

The route of the NN strategy is: $depot \rightarrow store4 \rightarrow store3 \rightarrow store2 \rightarrow store1 \rightarrow store5 \rightarrow store6$. The money lost by taking the route generated by MWLP model is 58.4, and the route's distance is about 41 km. The money lost by the NN strategy's route is 61.9, and its distance is about 43.1 km. Compared with the NN strategy, the supermarket chain not only reduces their loss but also shortens the truck's travel distance by taking the route given by the MWLP model. However, in some cases, the distance of the route given by MWLP can be longer. The MWLP model can guarantee minimizing the money lost during the transportation among all the models.

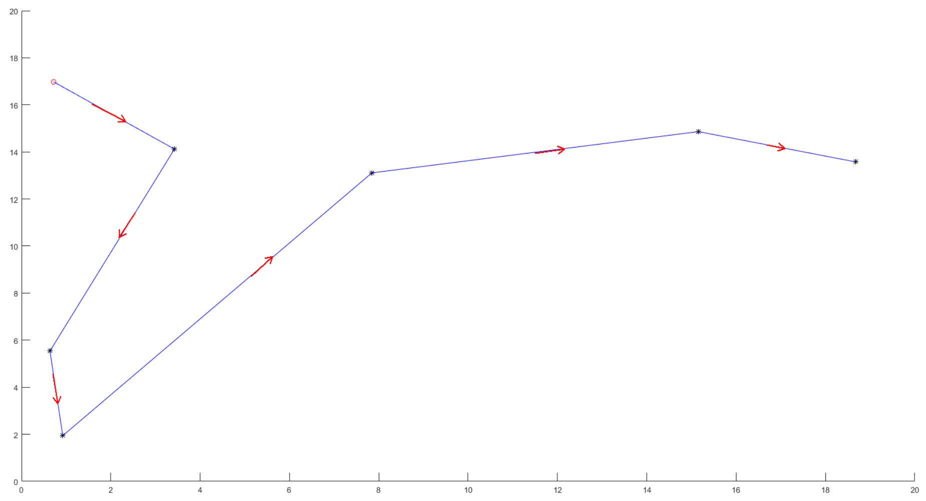

Figure 5.4: Vehicle route calculated by the MWLP model.

5.3 Conclusion and Future Work

In this chapter, we have described two real-world applications of MWLP. The instances show that MWLP is helpful when dealing with disaster relief and perishable commodity transportation problems.

However, the MWLP model still has shortcomings when it is used in real-world applications.

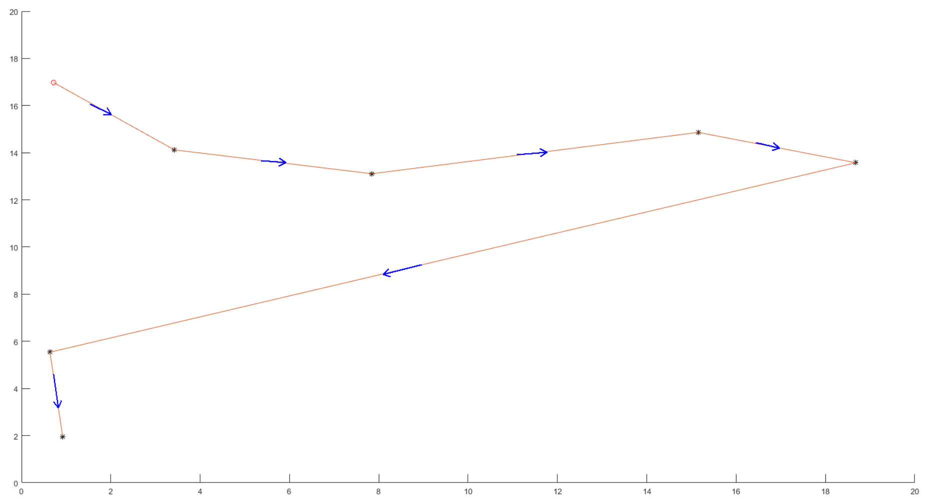

Figure 5.5: Vehicle route calculated by the NN strategy.

On the one hand, it only has one mobile agent. In most cases, people are dealing with situations with multiple mobile agents. On the other hand, both CRUT and LRUT are the slopes of linear functions. As a result, the real-world application must be modelled with a linear loss rate. This precondition is difficult to satisfy in most real situations. MWLP can be more powerful in real cases if these two limitations can be removed.

References

[1] G. Reinelt, *The Traveling Salesman*, vol. 840. Springer-Verlag, 1994.

[2] M. Dorigo and L. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[3] T. Stutzle and H. Hoos, "MAX-MIN Ant System and local search for the traveling salesman problem," in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pp. 309–314, IEEE, 1997.

[4] R. Gopal and D. van Gucht, "Genetic Algorithms for the Traveling Salesman Problem," in *Conference Paper*, pp. 160–168, 1985.

[5] J. Fakcharoenphol, C. Harrelson, and S. Rao, "The k-traveling repairmen problem," *ACM Transactions on Algorithms*, vol. 3, no. 4, pp. 40–es, 2007.

[6] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, "Paths, trees, and minimum latency tours," *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, vol. 2003-Janua, pp. 36–45, 2003.

[7] B. Y. Wu, Z. N. Huang, and F. J. Zhan, "Exact algorithms for the minimum latency problem," *Information Processing Letters*, vol. 92, no. 6, pp. 303–309, 2004.

[8] H. P. L. Luna, D. Problem, and T. S. Problem, "A New Flow Formulation for theMinimum Latency Problem," in *INOC - Internacional Network Optimization Conference*, vol. 81, p-p. 1–6, 1967.

[9] I. Ezzine, F. Semet, and H. Chabchoub, “New formulations for the traveling repairman problem,” in *8th International Conference of Modeling and Simulation*, no. 1976, pp. 8–13, 2010.

[10] H. B. Ban, K. Nguyen, M. Cuong Ngo, and D. N. Nguyen, “An efficient exact algorithm for the minimum latency problem,” in *Progress in Informatics*, no. 10, pp. 167–174, mar 2013.

[11] M. Goemans and J. M. Kleinberg, “An Improved Approximation Ratio for the Minimum Latency Problem,” *Mathematical Programming*, vol. 82, no. 1, pp. 111–124, 1998.

[12] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, “On the minimum latency problem,” in *Proc 26thSymp Theory of Computing STOC*, p. 9, 1994.

[13] A. Salehipour, K. Sörensen, P. Goos, and O. Bräysy, “Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem,” *4or*, vol. 9, no. 2, p-p. 189–209, 2011.

[14] T. A. Feo and M. G. Resende, “Greedy Randomized Adaptive Search Procedures,” *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[15] H. B. Ban and N. D. Nghia, “Improved Genetic Algorithm for Minimum Latency Problem,” in *Proceedings of the 2010 Symposium on Information and Communication Technology*, pp. 9–15, 2010.

[16] H. B. Ban and N. N. Duc, “A Meta-Heuristic algorithm combining between tabu and variable neighborhood search for the minimum latency problem,” *Fundamenta Informaticae*, vol. 156, no. 1, pp. 21–41, 2017.

[17] M. M. Silva, A. Subramanian, T. Vidal, and L. S. Ochi, "A simple and effective metaheuristic for the Minimum Latency Problem," *European Journal of Operational Research*, vol. 221, pp. 513–520, sep 2012.

[18] M. Moshref-Javadi and S. Lee, "A Taxonomy to the Class of Minimum Latency Problems," *Proceedings of the 2013 Industrial and Systems Engineering Research Conference*, vol. 144, pp. 3896–3905, 2013.

[19] E. Koutsoupias, C. Papadimitriou, and M. Yannakakis, "Searching a fixed graph," in *International Colloquium on Automata, Languages, and Programming*, pp. 280–289, 1996.

[20] B. Y. Wu, "Polynomial time algorithms for some minimum latency problems," *Information Processing Letters*, vol. 75, no. 5, pp. 225–229, 2000.

[21] R. Sitters, "The Minimum Latency Problem is NP-Hard for Weighted Trees," *Lecture Notes in Computer Science*, vol. 2337, pp. 230–239, 2002.

[22] T. Tulabandhula, C. Rudin, and P. Jaillet, "The machine learning and traveling repairman problem," in *International Conference on Algorithmic DecisionTheory*, pp. 262–276, Springer, 2011.

[23] M. Fischetti, G. Laporte, and S. Martello, "The Delivery Man Problem and Cumulative Matroids," *Operations Research*, vol. 41, no. 6, pp. 1055–1064, 1993.

[24] Y.-S. Liu, "Traveling Saleman problem with order delivery," Master's thesis, National Chiao Tung University, 2008.

[25] `https://class4bds.wordpress.com/2011/02/19/travelling-salesman-problem/`.

[26] J. Feng, “The Research of TSP’s Integer Programming modeling,” *China Management Informationization*, vol. 12, no. 23, pp. 63–67, 2009.

[27] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, vol. 57. Springer, 2003.

[28] J. J. Hopfield, “Artificial neural networks,” *IEEE Circuits and Devices Magazine*, vol. 4, no. 5, pp. 3–10, 1988.

[29] B. Yegnanarayana, *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[30] P. J. Angeline, P. J. Angeline, G. M. Saunders, G. M. Saunders, J. B. Pollack, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks.,” *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[31] Q. Zhang and H. Li, “MOEA/D: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[32] H. Raether, “Quantum computation with quantum dots,” *Phys. Rev. A*, vol. 57, no. 1, pp. 120–126, 1998.

[33] E. Knill, R. Laflamme, and G. Milburn, “A scheme for efficient quantum computation with linear optics.,” *Nature*, vol. 409, no. 6816, pp. 46–52, 2001.

[34] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[35] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, vol. 69. MIT press, 1992.

[36] J. D. Bastidas-Rodriguez, G. Petrone, C. A. Ramos-Paja, and G. Spagnuolo, “A genetic algorithm for identifying the single diode model parameters of a photovoltaic panel,” *Mathematics and Computers in Simulation*, vol. 131, pp. 38–54, 2014.

[37] M. Fogue, J. A. Sanguesa, F. Naranjo, J. Gallardo, P. Garrido, and F. J. Martinez, “Non-emergency patient transport services planning through genetic algorithms,” *Expert Systems with Applications*, vol. 61, pp. 262–271, 2016.

[38] X. Zeng, Y. Ge, J. Shen, L. Zeng, Z. Liu, and W. Liu, “The optimization of channels for a proton exchange membrane fuel cell applying genetic algorithm,” *International Journal of Heat and Mass Transfer*, vol. 105, pp. 81–89, 2017.

[39] V. F. Yu and S.-y. Lin, “A simulated annealing heuristic for the open location-routing problem,” *Computers & Operations Research*, vol. 62, pp. 184–196, 2015.

[40] M. P. V. S. Kirkpatrick, C. D. Gelatt Jr., “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[41] B. E. N. Goertzel and M. Ananda, “Simulated annealing on Uncorrelated energy landscapes,” *International Journal of Mathematics and Mathematical Sciences*, vol. 17, no. 4, pp. 791–798, 1994.

[42] G. B. Murali, B. B. V. L. Deepak, M. V. A. R. Bahubalendruni, B. B. Biswal, G. Bala Murali, B. B. V. L. Deepak, M. V. A. Raju Bahubalendruni, B. B. Biswal, G. B. Murali, B. B. V. L. Deepak, M. V. A. R. Bahubalendruni, and B. B. Biswal, “Optimal Assembly Sequence Planning Towards Design for Assembly Using Simulated Annealing Technique,” in *International Conference on Research into Design*, pp. 397–407, Springer, 2017.

[43] T. D. Kieu, "The travelling salesman problem and adiabatic quantum computation: An algorithm," *arXiv preprint arXiv:1801.07859*, 2018.

[44] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, 1995.

[45] H. S. Ramadan, A. F. Bendary, and S. Nagy, "Particle swarm optimization algorithm for capacitor allocation problem in distribution systems with wind turbine generators," *International Journal of Electrical Power and Energy Systems*, vol. 84, pp. 143–152, 2017.

[46] R. R. Trivedi, D. N. Pawaskar, and R. P. Shimpi, "Optimization of static and dynamic travel range of electrostatically driven microbeams using particle swarm optimization," *Advances in Engineering Software*, vol. 97, pp. 1–16, 2016.

[47] Z. Ye, Y. Ye, and H. Yin, "Qualitative and Quantitative Study of GAs and PSO Based Evolutionary Intelligence for Multilevel Thresholding," in *2017 10th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pp. 812–817, 2017.

[48] F. Glover, "Paths for Integer Programming," *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[49] K. Lenin, B. Ravindhranath Reddy, and M. Suryakalavathi, "Hybrid Tabu search-simulated annealing method to solve optimal reactive power problem," *International Journal of Electrical Power and Energy Systems*, vol. 82, pp. 87–91, 2016.

[50] M. R. Silva and C. B. Cunha, "A tabu search heuristic for the uncapacitated single allocation p-hub maximal covering problem," *European Journal of Operational Research*, vol. 262, no. 3, pp. 954–965, 2017.

[51] F. Ma, J. K. Hao, and Y. Wang, "An effective iterated tabu search for the maximum bisection problem," *Computers and Operations Research*, vol. 81, pp. 78–89, 2017.

[52] O. Shahvari and R. Logendran, "An Enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes," *Computers and Operations Research*, vol. 77, p-p. 154–176, 2017.

[53] M. Dorigo, V. Maniezzo, and a. Colorni, "The ant system: An autocatalytic optimizing process," *TR91-016, Politecnico di Milano*, pp. 1–21, 1991.

[54] K. Krynicki, M. E. Houle, and J. Jaen, "An efficient ant colony optimization strategy for the resolution of multi-class queries," *Knowledge-Based Systems*, vol. 105, pp. 96–106, 2016.

[55] Q. Yang, W. N. Chen, Z. Yu, T. Gu, Y. Li, H. Zhang, and J. Zhang, "Adaptive Multimodal Continuous Ant Colony Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 191–205, 2017.

[56] F. Zheng, A. Zecchin, J. Newman, H. Maier, and G. Dandy, "An Adaptive Convergence-Trajectory Controlled Ant Colony Optimization Algorithm with Application to Water Distribution System Design Problems," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 1–1, 2017.

[57] G. Păun, “Computing with membranes,” *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[58] T. Y. Nishida, “Membrane Algorithm With Brownian Subalgorithm and Genetic Subalgorithm,” *International Journal of Foundations of Computer Science*, vol. 18, no. 06, pp. 1353–1360, 2007.

[59] T. Y. Nishida, “Membrane algorithms,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3850 LNCS, pp. 55–66, 2006.

[60] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and F. J. Romero-Campero, “A uniform solution to SAT using membrane creation,” *Theoretical Computer Science*, vol. 371, no. 1-2, pp. 54–61, 2007.

[61] L. Pan, A. Alhazov, and T. O. Isdorj, “Further remarks on P systems with active membranes, separation, merging, and release rules,” *Soft Computing*, vol. 9, no. 9, pp. 686–690, 2005.

[62] V. Manca and L. Bianco, “Biological networks in metabolic P systems,” *BioSystems*, vol. 91, no. 3, pp. 489–498, 2008.

[63] F. J. Romero-Campero and M. J. Pérez-Jiménez, “Modelling gene expression control using P systems: The Lac Operon, a case study,” *BioSystems*, vol. 91, no. 3, pp. 438–457, 2008.

[64] M. J. Pérez-Jiménez and F. J. Romero-Campero, “Mechanisms, Symbols, and Models Underlying Cognition,” *Lecture Notes in Computer Science*, vol. 3561, pp. 268–278, 2005.

[65] B. Nagy and L. Szegedi, "Membrane computing and geographical operating systems," *Journal of Universal Computer Science*, vol. 12, no. 9, pp. 1312–1331, 2006.

[66] R. Ceterchi, R. Gramatovici, N. Jonoska, and K. Subramanian, "Tissue-like P Systems with Active Membranes for Picture Generation," *Fundamenta Informaticae*, vol. 56, no. 4, pp. 311–328, 2003.

[67] G. B. Enguix, "Unstable P systems: Applications to linguistics," in *Membrane Computing*, vol. 3365, pp. 190–209, 2004.

[68] G. Bel-Enguix and R. Gramatovici, "Parsing with Active P Automata," in *Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain,July, 17-22, 2003, Revised Papers*, vol. 2933, pp. 31–42, 2003.

[69] S. Yuan and X. Gu, "A Differential Evolution Based Membrane Computing," in *Shanghai Chemical and Chemical Society Annual Conference(SCCSAC)*, pp. 1–5, 2010.

[70] G. Păun and R. Păun, "Membrane Computing as a Framework for Modeling Economic Processes," in *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 1–22, 2005.

[71] G. Păun and R. Păun, "Membrane Computing and Economics: Numerical P Systems," *Fundamenta Informaticae*, vol. 73, no. 1,2, pp. 213–227, 2006.

[72] D. Guo, H. Xia, and Y. Zhou, "A New Optimised Algorithm for Solving Nonlinear Equations Based on Membrane Computing," *Computer Applications and Software(CAS)*, vol. 2, p. 42, 2013.

[73] M. À. Colomer, A. Margalida, D. Sanuy, and M. J. Pérez-Jiménez, “A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study,” *Ecological Modelling*, vol. 222, no. 1, pp. 33–47, 2011.

[74] B. Daniela, C. Paolo, P. Dario, and M. Giancarlo, “Seasonal variance in P system models for metapopulations,” *Progress in Natural Science*, vol. 17, no. 4, pp. 392–400, 2007.

[75] D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri, “Modelling metapopulations with stochastic membrane systems,” *BioSystems*, vol. 91, no. 3, pp. 499–514, 2008.

[76] G. X. Zhang, C. X. Liu, and H. N. Rong, “Analyzing radar emitter signals with membrane algorithms,” *Mathematical and Computer Modelling*, vol. 52, no. 11-12, pp. 1997–2010, 2010.

[77] A. Leporati and D. Pagani, “A membrane algorithm for the min storage problem,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4361 LNCS, pp. 443–462, 2006.

[78] J. Xiao, X. Zhang, and J. Xu, “A membrane evolutionary algorithm for DNA sequence design in DNA computing,” *Chinese Science Bulletin*, vol. 57, no. 6, pp. 698–706, 2012.

[79] T. Zhou, H. Z. Zhao, Y. Wang, and J. Yang, “A multi-vehicles optimization method for intersection cooperative driving,” *Proceedings of the 29th Chinese Control and Decision Conference, CCDC 2017*, pp. 5855–5861, 2017.

[80] L. Huang and N. Wang, “An Optimization Algorithm Inspired by,” in *International Conference on Advances in Natural Computation*, pp. 49–52, 2006.

[81] S. N. Krishna and R. Rama, “Breaking DES using P systems,” *Theoretical Computer Science*, vol. 299, no. 1-3, pp. 495–508, 2003.

[82] Y. Niu, S. Wang, J. He, and J. Xiao, “A novel membrane algorithm for capacitated vehicle routing problem,” *Soft Computing*, vol. 19, no. 2, pp. 471–482, 2014.

[83] A. Yan, H. Shao, and Z. Guo, “Weight optimization for case-based reasoning using membrane computing,” *Information Sciences*, vol. 287, pp. 109–120, 2014.

[84] W. Dong, K. Zhou, H. Qi, C. He, and J. Zhang, “A tissue P system based evolutionary algorithm for multi-objective VRPTW,” *Swarm and Evolutionary Computation*, no. November, pp. 1–13, 2017.

[85] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[86] B. Kim, J. Shim, and M. Zhang, “Comparison of TSP algorithms,” *Project for Models in Facilities Planning and . . .* , no. December, 1998.

[87] N. Mladenović and P. Hansen, “Variable neighborhood search,” *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[88] H. Dong, M. Huang, X. Wang, and B. Zheng, “Review of Variable Neighborhood Search Algorithm,” *Control Engineering of China*, vol. 7848, 2009.

[89] I. H. Osman and S. Ahmadi, “Guided construction search metaheuristics for the capacitated p-median problem with single source constraint,” *Journal of the Operational Research Society*, vol. 58, no. 1, pp. 100–114, 2007.

[90] H. Han, J. Ye, and Q. Lv, “A VNS-ANT algorithm to QAP,” *Proceedings - Third International Conference on Natural Computation, ICNC 2007*, vol. 3, no. Icnc, pp. 426–430, 2007.

[91] P. Hansen and N. Mladenović, “Variable neighborhood search: Principles and applications,” *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

[92] C. Avanthay, A. Hertz, and N. Zufferey, “A variable neighborhood search for graph coloring,” *European Journal of Operational Research*, vol. 151, no. 2, pp. 379–388, 2003.

[93] C. C. Ribeiro and M. C. Souza, “Variable neighborhood search for the degree-constrained minimum spanning tree problem,” *Discrete Applied Mathematics*, vol. 118, no. 1-2, pp. 43–54, 2002.

[94] K. Fleszar, I. H. Osman, and K. S. Hindi, “A variable neighbourhood search algorithm for the open vehicle routing problem,” *European Journal of Operational Research*, vol. 195, no. 3, pp. 803–809, 2009.

[95] A. Goel and V. Gruhn, “A General Vehicle Routing Problem,” *European Journal of Operational Research*, vol. 191, no. 3, pp. 650–660, 2008.

[96] J. Lazic, S. Hanafi, N. Mladenovic, and D. Urošević, “Solving 0-1 mixed integer programs with variable neighbourhood decomposition search,” *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 13, no. PART 1, pp. 2012–2017, 2009.

[97] B. Hu, M. Leitner, and G. R. Raidl, “Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem,” *Journal of Heuristics*, vol. 14, no. 5, pp. 473–499, 2008.

[98] K. Fleszar and K. S. Hindi, “Solving the resource-constrained project scheduling problem by a variable neighbourhood search,” *European Journal of Operational Research*, vol. 155, no. 2, pp. 402–413, 2004.

[99] E. K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman, “A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem,” *European Journal of Operational Research*, vol. 188, no. 2, pp. 330–341, 2008.

[100] M. Polacek, K. F. Doerner, R. F. Hartl, and V. Maniezzo, “A variable neighborhood search for the capacitated arc routing problem with intermediate facilities,” *Journal of Heuristics*, vol. 14, no. 5, pp. 405–423, 2008.

[101] J. Gao, L. Sun, and M. Gen, “A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems,” *Computers and Operations Research*, vol. 35, no. 9, pp. 2892–2907, 2008.

[102] C. J. Liao and C. C. Cheng, “A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date,” *Computers and Industrial Engineering*, vol. 52, no. 4, pp. 404–413, 2007.

[103] V. Bouffard and J. A. Ferland, “Improving simulated annealing with variable neighborhood search to solve the resource-constrained scheduling problem,” *Journal of Scheduling*, vol. 10, no. 6, pp. 375–386, 2007.

[104] N. Bjelić, M. Vidović, and D. Popović, “Variable neighborhood search algorithm for heterogeneous traveling repairmen problem with time windows,” *Expert Systems with Applications*, vol. 40, no. 15, pp. 5997–6006, 2013.

[105] P. Hansen, N. Mladenović, and J. A. Moreno Pérez, “Variable neighbourhood search: Methods and applications,” *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, 2010.

[106] J. Brimberg, P. Hansen, N. Mladenović, and E. D. Taillard, “Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem,” *Operations Research*, vol. 48, no. 3, pp. 444–460, 2000.

[107] Z. Wei and M. H. MacGregor, “Heuristic Approaches for Minimum Weighted Latency Problem,” *Icmmita 2016*, vol. 71, pp. 552–556, 2017.

[108] T. Takahashi, H. Nakagawa, Y. Satofuka, and K. Kawaike, “Flood and sediment disasters triggered by 1999 rainfall in Venezuela: A river restoration plan for an alluvial fan,” *Journal of natural disaster science*, vol. 23, no. 2, pp. 65–82, 2001.

[109] M. Ioualalen, J. Asavanant, N. Kaewbanjak, S. T. Grilli, J. T. Kirby, and P. Watts, “Modeling the 26 December 2004 Indian Ocean tsunami: Case study of impact in Thailand,” *Journal of Geophysical Research*, vol. 112, no. C7, p. C07024, 2007.

[110] M. Sajban, “A geological and geophysical context for the Wenchuan earthquake of 12,” *GSA today*, vol. 18, no. 7, p. 5, 2008.

[111] J. Holmes, Sir, “Lessons learned from the Haiti Earthquake Response - Humanitarian Practice Network,” *Humanitarian Exchange, Issue 48*, vol. 463, no. 48, pp. 2–3, 2010.

[112] “Line spacing in latex documents.” `http://www.10news.org`. Accessed June 4, 2017.

[113] M. Huang, K. R. Smilowitz, and B. Balcik, “A continuous approximation approach for assessment routing in disaster relief,” *Transportation Research Part B: Methodological*, vol. 50, pp. 20–41, 2013.

[114] L. E. De la Torre, I. S. Dolinskaya, and K. R. Smilowitz, “Disaster relief routing: Integrating research and practice,” *Socio-Economic Planning Sciences*, vol. 46, no. 1, pp. 88–97, 2012.

[115] M. Huang, K. Smilowitz, and B. Balcik, “Models for relief routing: Equity, efficiency and efficacy,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 2–18, 2012.

[116] S. Lee, “The role of centrality in ambulance dispatching,” *Decision Support Systems*, vol. 54, no. 1, pp. 282–291, 2012.

Appendix

In Chapter 3 and Chapter 4, the experiments' outputs are shown in figures. Most of the figures are histograms which are great for comparison. They are able to show the advantages and shortages of different algorithms in a very clear way. However, this method may result in a loss of precision. In this appendix, I list the exact numbers of all the experiments as tables.

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	603.2215	677.8057	593.3672	20.8758	0.5547
SA:	677.3265	827.7803	593.3672	50.9349	0.0281
PSO:	600.0853	636.9034	593.3672	8.6668	0.9063
TS:	671.8106	804.1919	593.3672	53.9989	0.1676
ACO:	601.2590	603.4848	593.3672	4.2123	0.2459
Iteration Number 50:					
GA:	600.6858	677.8057	593.3672	16.1153	0.9123
SA:	670.3135	782.5323	593.3672	48.8225	0.0463
PSO:	598.1100	611.1832	593.3672	5.4997	1.5055
TS:	671.7441	824.9448	593.3672	57.9242	0.2777
ACO:	600.5507	603.4848	593.3672	4.6141	0.4106
Iteration Number 100:					
GA:	602.4315	677.8057	593.3672	18.3249	1.8141
SA:	654.2437	773.7990	593.3672	49.1445	0.0903
PSO:	599.1615	607.7651	593.3672	5.2126	3.0012
TS:	672.0896	824.9448	593.3672	53.4221	0.5584
ACO:	598.9319	603.4848	593.3672	5.0588	0.8155

Table 5.1: Experiment results for the instance with 12 vertices

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	869.0974	908.3766	777.4809	22.0301	0.5686
SA:	916.4789	1211.1790	777.4809	112.0694	0.0307
PSO:	932.4069	1101.5480	777.4809	76.4353	1.0000
TS:	921.4440	1349.1370	777.4809	121.9219	0.1752
ACO:	810.9843	853.8841	779.3238	18.2544	0.3719
Iteration Number 50:					
GA:	853.5262	885.7933	777.4809	17.7395	0.9099
SA:	873.6359	1164.7730	777.4809	87.1309	0.0495
PSO:	888.5413	1092.0090	793.0355	44.1926	1.7027
TS:	912.4278	1278.1990	777.4809	124.6073	0.2918
ACO:	804.6404	840.5190	779.2319	14.4444	0.6105
Iteration Number 100:					
GA:	853.1107	885.7933	777.4809	14.8879	1.7821
SA:	865.4213	1228.5790	777.4809	88.8633	0.0970
PSO:	855.6312	926.8336	782.6950	33.3183	3.3809
TS:	891.0962	1278.1990	777.4809	104.8750	0.5848
ACO:	799.2202	831.9296	779.2319	14.0389	1.2198

Table 5.2: Experiment results for the instance with 16 vertices

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	1260.3200	1419.0640	1137.3980	56.7416	0.5421
SA:	1362.5740	1760.4110	1135.0280	154.7496	0.0316
PSO:	1531.7590	1848.0160	1204.9420	144.9686	1.1205
TS:	1347.2510	1830.2990	1104.8100	159.7022	0.2184
ACO:	1117.6580	1143.2240	1104.8100	10.2207	0.4790
Iteration Number 50:					
GA:	1156.9450	1261.2230	1104.8100	34.1416	0.9166
SA:	1363.4870	1735.8520	1118.4730	150.9704	0.0505
PSO:	1438.8230	1759.8930	1164.6990	127.2034	1.8607
TS:	1313.3120	1710.5990	1104.8100	142.2239	0.3613
ACO:	1111.8600	1135.5230	1104.8100	8.8050	0.7936
Iteration Number 100:					
GA:	1147.6420	1282.1820	1104.8100	35.3245	1.7850
SA:	1303.3580	1705.9430	1104.8100	136.8670	0.0990
PSO:	1338.4110	1624.7380	1126.8150	103.1663	3.6755
TS:	1310.7600	1655.2170	1104.8100	132.4501	0.7143
ACO:	1110.6500	1127.3740	1104.8100	8.2312	1.5839

Table 5.3: Experiment results for the instance with 20 vertices with uniform weighted distribution

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	1626.6090	1632.7950	1598.7180	6.0662	0.5319
SA:	1794.2700	2253.9790	1528.1610	165.0535	0.0290
PSO:	1915.6850	2413.6300	1592.7740	138.7305	1.0298
TS:	1719.8710	2257.1890	1528.1610	161.5153	0.1948
ACO:	1580.8280	1623.4080	1551.9490	18.3299	0.4563
Iteration Number 50:					
GA:	1626.7140	1632.7950	1598.7180	5.7281	0.8803
SA:	1818.6200	2284.9950	1552.6630	171.8900	0.0469
PSO:	1799.5240	2082.3750	1575.8200	122.8223	1.7088
TS:	1736.1720	2199.4080	1528.1610	168.0478	0.3256
ACO:	1565.2530	1606.8960	1531.1210	14.3178	0.7583
Iteration Number 100:					
GA:	1624.4280	1632.7950	1531.1210	14.2001	1.7537
SA:	1712.9770	2042.5580	1528.1610	124.8291	0.0924
PSO:	1686.6620	1905.7020	1531.1210	82.1366	3.3977
TS:	1709.3860	2105.2760	1528.1610	141.6178	0.6499
ACO:	1535.1240	1555.7190	1531.1210	6.3860	1.5075

Table 5.4: Experiment results for the instance with 20 vertices with randomly weighted distribution

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	5678.88	6268.00	5209.44	222.61	0.58
SA:	6876.94	8823.85	5445.00	769.32	0.03
PSO:	9905.13	13685.76	7784.53	1036.18	1.39
TS:	6089.56	8541.06	5155.49	612.44	0.30
ACO:	5473.05	5751.70	5274.82	86.28	0.80
Iteration Number 50:					
GA:	5508.10	5955.83	5209.44	147.34	0.95
SA:	6639.20	8078.47	5323.72	661.56	0.05
PSO:	9113.20	10694.52	7367.37	860.24	2.29
TS:	5951.91	7453.70	5147.46	497.51	0.50
ACO:	5417.24	5561.88	5248.01	66.01	1.32
Iteration Number 100:					
GA:	5477.54	5897.52	5109.95	127.38	1.87
SA:	6396.33	7905.16	5182.79	645.54	0.10
PSO:	8104.93	9884.62	6508.74	725.32	4.57
TS:	5871.55	7449.99	5089.24	485.76	1.00
ACO:	5327.76	5392.92	5209.44	43.61	2.65

Table 5.5: Experiment results for the instance with 30 vertices

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	8512.7510	8857.5780	8238.3010	207.2454	0.5881
SA:	10627.9700	12886.6100	9071.3120	811.1912	0.0351
PSO:	18721.4200	21634.2500	15637.3300	1502.2850	1.8398
TS:	9277.4980	11570.6100	8172.8470	658.0425	0.4311
ACO:	7942.8000	8192.1610	7631.5290	109.6624	1.1468
Iteration Number 50:					
GA:	8417.2780	8768.6450	8207.0250	177.4348	0.9752
SA:	10178.3400	11977.4800	8201.9290	734.0437	0.0569
PSO:	17492.1200	21394.1100	14480.8000	1404.8780	3.0807
TS:	8953.0470	10363.1300	7869.6110	584.7266	0.7111
ACO:	7828.2380	8113.7690	7490.8510	118.3647	1.8924
Iteration Number 100:					
GA:	8353.6770	8564.1840	8141.7140	113.8208	1.9207
SA:	9934.0700	11717.6300	7902.6340	875.6051	0.1117
PSO:	15324.1600	19457.6500	12242.7100	1443.6570	6.1287
TS:	8859.0920	10364.8500	7789.6710	604.3225	1.4307
ACO:	7687.5530	7849.7550	7533.4330	70.6487	3.8650

Table 5.6: Experiment results for the instance with 40 vertices

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	14748.3400	15092.8300	13965.3500	247.9989	0.6039
SA:	18729.2100	22559.2800	15527.2800	1359.3280	0.0385
PSO:	32413.3200	38511.4300	27231.1300	2141.4860	2.4648
TS:	15489.5600	18277.1900	13478.0200	901.5592	0.5850
ACO:	13282.7400	13957.3700	12408.2100	260.8826	1.5673
Iteration Number 50:					
GA:	14416.6000	14919.5700	13541.8900	333.0463	0.9908
SA:	17612.5000	20912.5300	14683.4700	1284.0710	0.0622
PSO:	29955.6900	37545.5000	24540.1700	2637.1680	4.0649
TS:	14787.3300	16644.2400	13314.2100	737.5127	0.9652
ACO:	13074.3800	13475.8200	12459.6400	226.1355	2.5905
Iteration Number 100:					
GA:	13819.9800	14842.3400	12868.6200	471.0468	1.9844
SA:	16833.5700	20574.9900	13460.0800	1186.5610	0.1213
PSO:	26392.8300	33295.4000	20089.7500	2547.3040	8.0708
TS:	14581.8500	16206.2600	13216.5000	644.1055	1.9513
ACO:	12472.5800	12763.8600	12195.5700	110.0858	5.1760

Table 5.7: Experiment results for the instance with 50 vertices

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 30:					
GA:	87627.20	90337.09	82442.53	1815.97	0.73
SA:	147811.70	166967.80	120740.00	9644.16	0.06
PSO:	352055.10	407250.30	295119.10	22217.54	6.99
TS:	100208.70	115845.90	82337.77	6873.94	1.64
ACO:	83037.29	86364.09	79360.62	1580.93	4.45
Iteration Number 50:					
GA:	85416.67	89734.32	80277.71	2203.74	1.21
SA:	133627.70	154046.40	114187.80	8839.96	0.09
PSO:	332587.60	378245.10	289756.20	19814.25	11.59
TS:	95959.24	109807.70	82414.01	6043.65	2.73
ACO:	82305.71	85506.49	77471.84	1385.70	7.40
Iteration Number 100:					
GA:	81955.25	88002.88	76412.31	2125.15	2.40
SA:	119914.10	146227.90	96907.38	8527.05	0.18
PSO:	301642.00	340973.90	255877.10	18313.65	23.06
TS:	92914.09	108651.10	81777.86	6064.40	5.47
ACO:	78474.57	80792.17	74990.68	1294.67	14.86

Table 5.8: Experiment results for the instance with 100 vertices

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 100:	6396.33	7905.16	5182.79	645.54	0.10
Iteration Number 200:	6089.33	7655.79	5089.24	478.83	0.21
Iteration Number 300:	5977.45	7410.86	5230.21	491.31	0.31
Iteration Number 500:	5879.41	7879.25	5089.24	510.48	0.51

Table 5.9: Experiment results for the instance with 30 vertices for the SA method with different iterations

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 100:	16833.57	20574.99	13460.08	1186.56	0.12
Iteration Number 200:	16274.70	19311.88	13925.50	1129.99	0.24
Iteration Number 300:	15778.70	19850.15	13464.15	1080.12	0.36
Iteration Number 500:	15435.06	18217.95	12799.65	1002.67	0.59

Table 5.10: Experiment results for the instance with 50 vertices for the SA method with different iterations

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Iteration Number 100:	119914.10	146227.90	96907.38	8527.05	0.18
Iteration Number 200:	112737.10	132484.50	92907.76	7591.20	0.35
Iteration Number 300:	109564.70	128814.40	89164.08	7787.23	0.52
Iteration Number 500:	104423.50	128507.40	84340.42	8072.78	0.87

Table 5.11: Experiment results for the instance with 100 vertices for the SA method with different iterations

	Average Value	Max Value	Min Value
ACO:	12472.58	12763.86	12195.57
AC_Local:	13870.40	15973.55	12469.34

Table 5.12: Comparison of results for two ACO algorithms. Correspond to Figure 3.14

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 20:	1110.65	1104.81	1127.37	8.23	1.58
Vertex Number 30:	5327.76	5209.44	5392.92	43.61	2.65
Vertex Number 50:	12472.58	12195.57	12763.86	110.09	5.18
Vertex Number 100:	34087.37	32543.18	35263.06	494.67	15.00

Table 5.13: Experiment results for ACO running the test instances for new heuristics, corresponding for Figure 4.5 to Figure 4.8

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 20:	1104.81	1104.81	1104.81	0	20.88
Vertex Number 30:	5194.75	5089.24	5305.45	66.57	28.11
Vertex Number 50:	12297.16	12051.85	12757.56	146.88	46.96
Vertex Number 100:	31935.91	30768.41	33100.52	411.89	119.77

Table 5.14: Experiment results for the algorithm of P System running in sequential

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 20:	1104.81	1104.81	1104.81	0	13.39
Vertex Number 30:	5230.40	5089.24	5365.99	68.90	18.82
Vertex Number 50:	12296.86	12037.95	12830.64	168.65	34.76
Vertex Number 100:	31918.81	31220.30	33071.37	385.95	95.41

Table 5.15: Experiment results for the algorithm of P System running in parallel

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 20:	1104.81	1104.81	1104.81	0	32.20
Vertex Number 30:	5236.78	5139.27	5287.45	30.90	133.98
Vertex Number 50:	12430.55	12110.70	12674.19	92.72	270.06
Vertex Number 100:	34090.67	32863.00	34677.42	348.77	1508.45

Table 5.16: Experiment results for the algorithm of VND based on ACO2 running in sequential

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 20:	1104.81	1104.81	1104.81	0	5.59
Vertex Number 30:	5235.51	5153.51	5288.03	32.76	21.68
Vertex Number 50:	12425.76	12206.42	12609.79	86.53	47.80
Vertex Number 100:	34030.27	32059.98	34877.54	440.05	246.20

Table 5.17: Experiment results for the algorithm of VND based on ACO2 running in parallel

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 20:	1104.81	1104.81	1104.81	0	3.29
Vertex Number 30:	5261.42	5109.95	5322.59	41.59	29.75
Vertex Number 50:	12386.70	12144.32	12588.29	91.04	52.46
Vertex Number 100:	33123.00	31591.22	34086.87	482.83	102.95

Table 5.18: Experiment results for the algorithm of VND based on ACO1 running in sequential

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 20:	1104.81	1104.81	1104.81	0	0.46
Vertex Number 30:	5261.42	5121.31	5315.30	37.61	2.89
Vertex Number 50:	12391.44	12174.16	12585.98	88.22	5.01
Vertex Number 100:	33029.69	31932.28	34069.08	475.91	11.03

Table 5.19: Experiment results for the algorithm of VND based on ACO1 running in parallel

	Average Value	Max Value	Min Value	Standard Deviation	Running Time
Vertex Number 30:	6471.82	7509.74	5413.27	485.03	1.22e-04
Vertex Number 50:	16227.11	19316.47	13831.50	1078.50	5.08e-04
Vertex Number 100:	968533.60	118179.50	863934.40	6257.26	1.26e-03

Table 5.20: Experiment results for ILS