Université Lille Nord de France
Pôle de Recherche et d'Enseignement Supérieur

N. d'ordre : 12 - 0546

Université Lille Nord de France | Université d'Artois

École doctorale SPI Lille Nord de France

THÈSE DE DOCTORAT DE L'UNIVERSITÉ D'ARTOIS

(Spécialité: Génie Informatique et Automatique)

Présentée par

M. Yuhan GUO

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ D'ARTOIS

Sujet de la thèse :

Metaheuristics for Solving Large Size Long-term Car Pooling Problem and an Extension

Métaheuristiques pour la Résolution de Problème de Covoiturage Régulier de Grande Taille et d'une Extension

Soutenue le 09 novembre 2012 devant le jury composé de:

Rapporteurs:	M. Van-Dat CUNG	Professeur, Institut Polytechnique de Grenoble
	M. Jin-Kao HAO	Professeur, Université d'Angers
Examinateurs:	Mme. Marie-José HUGUET	MCF, HDR, Institut National des Sciences Appliquées de Toulouse
	M. El-Ghazali TALBI	Professeur, Université des Sciences et Technologies de Lille
Directeur de thèse:	M. Gilles GONCALVES	Professeur, Université d'Artois
Co-encadreur:	M. Tienté HSU	MCF, Université d'Artois

LGI2A Laboratoire de Génie Informatique et d'Automatique de l'Artois - EA 3926

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, I am heartily thankful to my supervisor, Prof. Gilles Goncalves, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject and to finish this dissertation. Without his consistent and illuminating instruction, the completion of this thesis would not have been possible.

I am also sincerely grateful to my co-supervisor, Dr. Tienté Hsu, for the support and guidance he showed me throughout my study. His instructive advice and useful suggestions are very valuable to my research.

I would like to extend my sincere gratitude to Prof. Van-Dat Cung, Dr. Marie- José Huguet, Prof. Jin-Kao Hao, and Prof. El-Ghazali Talbi, for serving on my thesis committee and for the time and the effort given to examine this dissertation.

Acknowledgements are also made to all the members of LGI2A laboratory for their help and encouragement. I am very lucky to have them as my colleagues.

Last but not the least, my thanks go to my beloved family for their loving considerations and great confidence in me all through these years, it has been really important to me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

Nowadays, the increased human mobility combined with high use of private cars increases the load on environment and raises issues about quality of life. The extensive use of private cars lends to high levels of air pollution, parking problem, traffic congestion and low transfer velocity. In order to ease these shortcomings, the car pooling program, where sets of car owners having the same travel destination share their vehicles, has emerged all around the world.

In the beginning of 20th century, the widespread use of internet and mobile phones has greatly helped car pooling to expand by enabling people to find, contact and arrange their car pool members more easily. However, the car pooling shows a lack of research on its optimization, since only very few works can be found in the literature. With such background, we present here our research on the long-term car pooling problem. In this thesis, the long-term car pooling problem is modeled and metaheuristics for solving the problem are investigated.

The thesis is organized as follows. First, the definition and description of the problem as well as its mathematical model are introduced. Then, several metaheuristics to effectively and efficiently solve the problem are presented. These approaches include a Variable Neighborhood Search Algorithm, a Clustering Ant Colony Algorithm, a Guided Genetic Algorithm and a Multi-agent Self-adaptive Genetic Algorithm. Experiments have been conducted to demonstrate the effectiveness of these approaches on solving the long-term car pooling problem. Afterwards, we extend our research to a multi-destination daily car pooling problem, which is introduced in detail manner along with its resolution method. At last, an algorithm test and analysis platform for evaluating the algorithms and a car pooling platform designed for the students of Artois University are presented in the appendix.

Key-words: optimization, car pooling problem, local search, metaheuristic, hyper-heuristic, multi-agent system.

Résumé

La dispersion spatiale de l'habitat et des activités de ces dernières décennies a fortement contribué à un allongement des distances et des temps de trajets domicile-travail. Cela a pour conséquence un accroissement de l'utilisation des voitures particulières, notamment au sein et aux abords des grandes agglomérations. Ce boom de la mobilité n'est pas sans conséquence et l'actualité nous le rappelle chaque jour: la pollution atmosphérique, les accidents de la route, les embouteillages, … Afin de réduire les impacts dus à l'augmentation du trafic routier, des services de covoiturage, où des usagers ayant la même destination se regroupent en équipage (un chauffeur et des passagers) pour se déplacer, ont été mis en place partout dans le monde.

Les avancées technologiques de ces dernières décennies de l'internet, de la téléphonie mobile et des systèmes de géolocalisation ont largement contribué à faciliter l'utilisation des systèmes de covoiturage, elles permettent plus facilement aux usagers de trouver, de contacter et de constituer les équipages. Toutefois, le problème du covoiturage semble être le parent pauvre de la famille des problèmes de tournées de véhicules: très peu de travaux concernent cette problématique. Néanmoins, on constate depuis quelques années, un intérêt accru pour ce problème, dû aux mouvements écologiques, à l'augmentation des prix du carburant, … Nous présentons ici nos travaux sur le problème de covoiturage régulier. Dans cette thèse, le problèmes de covoiturage régulier a été modélisé et plusieurs métaheuristiques de résolution ont été implémentées, testées et comparées.

La thèse est organisée de la façon suivante: tout d'abord, nous commençons par présenter la définition et la description du problème ainsi que le modèle mathématique associé. Ensuite, plusieurs métaheuristiques pour résoudre le problème sont présentées. Ces approches sont au nombre de quatre: un algorithme de recherche locale à voisinage variable, un algorithme à base de colonies de fourmis, un algorithme génétique guidée et un système multi-agents génétiques auto-adaptatif. Des expériences ont été menées pour démontrer l'efficacité de nos approches. Nous continuons ensuite avec la présentation et la résolution d'une extension du problème de covoiturage occasionel comportant plusieurs destinations. Pour terminer, une plate-forme java de test et d'analyse pour évaluer nos approches et une plate-forme de covoiturage conçue pour les étudiants de l'Université d'Artois sont présentées dans l'annexe.

Mots-clés: optimisation, problème du covoiturage, recherche locale, métaheuristique, hyper-heuristique, système multi-agents.

Introduction

This Ph.D. thesis focuses on solving complex combinatorial problems and particularly logistic and transport problems. It presents the results of three years' research held in laboratory LGI2A of Artois University in France.

Rising vehicle number and increased use of private cars have caused significant traffic congestion, noise and energy waste. Public transport cannot always be set up in the non-urban areas. Car pooling, which is based on the idea that sets of car owners having the same travel destination share their vehicles, has emerged to be a viable possibility for reducing private car usage around the world.

Nowadays, more and more information and communication systems become available to serve the real-world applications. The widespread use of World Wide Web, Geographic Information Systems (GIS), Global Positioning Systems (GPS), and mobile telephones makes the car pooling programs more and more popular and easy-to-implement. In spite of the interest in real-world applications, the research on the optimization of the car pooling problem is still limited. In the literature, only very few studies can be found for the car pooling problem. Especially, there is a lack of research for the long-term car pooling problem, which is however commonly used by large companies, public organizations and universities.

During the last decade, metaheuristics are raising a large interest in optimization community and particularly in the transportation domain. They represent more general approximate algorithms applicable to a large variety of optimization problems. They provide acceptable solutions in a reasonable time for solving hard and complex problems in science and engineering. The instances of problems are solved by exploring a large solution search space, and the metaheuristics achieve this by reducing the effective size of the space and by exploring that space efficiently.

In this thesis, our research explores the use of metaheuristics to solve the long-term car pooling problem (LTCPP). In the LTCPP, each user has to act as both a server and a client and a solution is to define user pools where each user will in turn, on different days, pick up the remaining pool members. The objective is to minimize the amount of vehicles used and the total distance traveled by all users, subject to car capacity and time window constraints. The LTCPP can be considered as a combination of a clustering problem and a routing problem. It requires finding the car pool members relatively close to each other and identifying the route and schedule for each member in the car pool with respect to the car capacity and time window constraints. The long-term car pooling problem is NP-complete which indicates the high complexity in solving the problem. Moreover, in real world application, large companies or universities usually have thousands of participants for a car pooling program, which provides large instances to solve. Although the long-term car pooling program focuses

on a long-term scheduling, the users in the real-world application always require their schedule to be generated in a short period of time, so they can examine the schedule, find the unsatisfactory and inconvenience, and then submit their requests of modification. As a result, the organizer needs to respond to the users as soon as possible. Thus, the LTCPP requires fast and accurate algorithms to solve the instances, since nowadays only zero-delay systems can attract users and survive in the competition with all their rivals.

However, along with the lack of solving methods designed for the long-term car pooling problem, the existing approaches are either time consuming or lacking of solution quality for the real-world application. Among all the current literature found, none of the methods developed is cost-effective enough when dealing with large scale instances. Thus, more efficient and powerful meta-heuristics are still required to meet the practical requirements. These new approaches should be faster, easier to use and more robust. The goal of our current work is aimed at developing such metaheuristics to generate solutions with good quality for large-scale real-world applications.

The main goal of our research is listed as follows.

- Create an accurate mathematical model for the long-term car pooling problem with respect to the real-world situation.
- Develop metaheuristics which provide high solution quality.
- Develop metaheuristics which can solve large instances efficiently.
- Develop metaheuristics which ease the implementation in real-world.
- Develop a car pooling platform for the students of the Artois University.

The research presented in this thesis has progressed in three phases. In the first phase, state-of-art that covers a description and specificities of the problem is given. The solving methods for the problem and for the related problems are presented in order to provide a global view of the researches carried out in this domain. Besides, the benchmarks of the problem and their development are reported.

The second phase, which is the main phase of our research, introduces the development of algorithms for the LTCPP. These algorithms cover the different classes of metaheuristics.

Metaheuristics can be divided into two categories: trajectory-based metaheuristics and population-based metaheuristics. The main difference of these two kinds of methods relies in the number of tentative solutions used in each step of the iterative algorithm. A trajectory-based technique starts with a single initial solution. At each step of the search, the current solution is replaced by another solution found in its neighborhood. It is usual that trajectory-based metaheuristics allow quickly finding a locally optimal solution, and so they are called exploitation-oriented methods promoting intensification in the search space. On the other hand, population-based algorithms make use of a population of solutions. The initial population is normally randomly generated or created with a cheap algorithm, and then enhanced through an iterative process. At each generation of the process, the whole or a part of population

is replaced by newly generated individuals. These techniques are called exploration-oriented methods, since their main ability resides in the diversification in the search space.

Our first attempt to solve the LTCPP with metaheuristics is by using trajectory-based metaheuristics. The most recent and popular approach in the field of trajectory-based metaheuristics is Variable Neighborhood Search (VNS). Its basic idea is systematic change of neighborhood within a local search in order to get different local optima and to escape from the current local optimum. In addition, the characteristic of changing the neighborhood structure might offer a higher probability in finding the global optimum. Since different neighborhoods generate different landscapes, a solution that is locally optimal on the search landscape with respect to a neighborhood is probably not locally optimal with respect to another one.

For that purpose, we address in this thesis the class of trajectory-based metaheuristics represented by a Variable Neighborhood Search particularly designed for the LTCPP.

As abovementioned, the trajectory-based metaheuristics manage only one solution in each iteration. The solution obtained in each iteration is based on the previous iteration. This mechanism provides the trajectory-based metaheuristics good intensification search ability; however, they are limited in diversity. Moreover, some good characteristics from the former solutions are lost since only one solution is conserved to the next iteration. Therefore, an efficient and effective algorithm should not only be able to focus on improving the current solution, but also be able to maintain a good diversity and memorize the good composition of the former solutions in the search process.

Population-based metaheuristics provide a number of potential advantages for such purposes. They start from an initial population of solutions and iteratively generate a new population based on the current population, and then replace the current one with the new one. The new population can maintain some useful characteristics of the old population, so the fine solutions of previous iterations are always inherited and the solution quality of the population are improved.

There are two main families in the population-based metaheuristics: swarm intelligence metaheuristics and evolutionary metaheuristics. The swarm intelligence metaheuristics are typically made up of a population of simple agents interacting locally with each other and with their environment. The inspiration often comes from nature. The agents follow very simple rules, and there is no centralized control structure dictating how individual agents should behave, but the interactions between such agents lead to the emergence of an intelligent global behavior. The evolutionary metaheuristics use some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines which individuals survive to the next generation. Evolution of the population takes place with repeated applications of the above procedure.

Based on the characteristics of the LTCPP, both the two families are considered interesting to be applied to solve our problem. In the swarm intelligence family, we select the most commonly used algorithm structure, Ant Colony Optimization, since it has been proven to be an effective solver for a broad range of transportation problems, and more important, its good ability in constructing path in a graph encourages their use for our problem, since identifying the routes for users is an importance phase in the LTCPP.

For this purpose, we present in this thesis a Clustering Ant Colony Algorithm for solving the long-term car pooling problem. The algorithm is based on the Ant Colony Optimization paradigm. A preference mechanism is designed to merge the clustering and routing operations together in order to gain a good ability to obtain high solution quality. The approach is proven to be able to track high quality solutions in the search space.

In the evolutionary algorithm family, our selection is the Genetic Algorithm. The Genetic Algorithm does not appear to have made a great impact so far on the car pooling problem, but grounding on the specification of the LTCPP, which is a combination of clustering and routing, we believe Genetic Algorithm is a suitable paradigm for solving this problem on the basis of its good exploration ability and flexible chromosome representation.

Therefore, we developed a Guided Genetic Algorithm for solving the long-term car pooling problem. In the Guided Genetic Algorithm, the composition of the better individuals will always be memorized and updated. Then this information will be used for guiding the genetic operators, in order to produce more feasible offspring solutions with high solution quality. Moreover, an adaptive parameter control is designed to maintain the balance between the intensity and the diversity of the search process.

Although the two population-based metaheuristics are proven to be able to provide good solution quality, some weaknesses in solving the LTCPP appears during our research. First, although the use of metaheuristics allows to significantly reducing the computational complexity of the search process, the latter remains time or memory consuming for the large size instances. Second, the algorithm's ability to explore other areas of the search space is significant decreased after the convergence to an optimum. Third, the population-based algorithms require a large number of accurate parameter settings in order to obtain good search ability, thus a complex and time consuming parameter testing phase is required. At last, the structures of the algorithms are always fixed, thus the new operators or constraints are hard to insert into or remove from the system without modifying the algorithm structure. Therefore, an improved approach for solving the LTCPP is required.

This can be achieved by a multi-agent system with hyper-heuristic. Multi-agent systems is a subfield of Artificial Intelligence research dedicated to the development of distributed solutions to complex problems regarded as requiring intelligence. It is designed to improve the computational speed and to maintain the diversity after the convergence by communicating among the agents. The hyper-heuristic is defined as

using heuristics to choose heuristics. The fundamental difference between metaheuristics and hyper-heuristics is that most implementations of metaheuristics search within a search space of problem solutions, whereas hyper-heuristics always search within a search space of heuristics. Thus, hyper-heuristics are used to find the most suitable heuristic or sequence of heuristics in a given situation, so the design of each individual heuristic becomes more flexible. Furthermore, with the hyper-heuristic, any new operator can be easily inserted into the system without modifying the system's structure, since the hyper-heuristic will select the most appropriate heuristic to apply.

Thus, we investigate in this thesis to merge a population-based metaheuristic with the multi-agent system and the hyper-heuristic. For this purpose, we elaborate a Multi-agent Self-adaptive Genetic Algorithm for solving the long-term car pooling problem.

In the last phase of our research, we extend our work to the daily car pooling problem (DCPP). In the DCPP, a number of users declare their availability for picking up or bringing back other users on one particular day. Hence, these users are considered as servers, and the other users being picked up or bringing back are considered as clients. Then the problem becomes to assign clients to servers and to identify the routes to be driven by the servers. According to our observation to the daily car pooling applications, we realized a fact that users going to different destinations normally are separated into different car pool projects even they live in the same neighborhood. In reality, different car pooling projects may have the destinations close to each other, but the current daily car pooling program will divide users according to their destinations, only the users going to the same destination are pooled together, even a lot of servers travel pass other destinations before reaching their own destinations with an empty car. Servers are not able to pick up their neighbors because the neighbors go to different destinations, even these destinations will be passed by the servers during their journey. This situation greatly decreases the effectiveness of serving the users and potentially increased the travel cost of all the participants in the daily car pooling project, since if a server can pick up other clients who go to the destinations other than the server's own one, the total travel cost can be greatly decreased.

Thus, a new daily car pool model which includes multiple destinations in one program is defined in this thesis. The server in multi-destination daily car pooling can pick up clients who go to different destinations as long as the server can accept the length of the detour he/she has to make. Two servers in the model can be given a transfer point, where the clients can change vehicles in order to reach their destinations in time and avoid the server to make long detours.

A resolution approach is also designed for the MDCPP. The method is a hybrid approach based on the Ant Colony Optimization paradigm. Experiments are performed to confirm the ability of the approach in solving the MDCPP.

During our research, two platforms are developed. The first platform is designed for the test, demonstration, evaluation and comparison among all the approaches for solving the LTCPP. The parameters, solutions and result evaluations can be viewed in

a graphical interface, which aids and facilitates all the operations. The second platform is a web based application designed for car pooling service of the students of Artois University. The car pooling participants can submit their requests and the platform generate the routes and schedules by using the metaheuristics introduced in this thesis. The platform integrates Google Map API, so the routes for the participants can be tracked and viewed graphically.

All abovementioned aspects are addressed in this thesis, providing a holistic view on the challenges and opportunities of applying metaheuristics to the car pooling problem, and suitable novel approaches are developed for each aspect.

The structure of this thesis is described as follows:

• Chapter 1 gives a general overview on the long-term car pooling problem. The mathematical model for the problem is presented. Afterwards, existing approaches to the problem are introduced, as well as the resolution approaches to the problems related to the long-term car pooling problem. Finally, the benchmark sets used in our experimentations are introduced.

• Chapter 2 deals with solving the long-term car pooling problem with trajectory-based metaheuristics. It begins with the common concepts of this class of metaheuristics. Then, a Variable Neighborhood Search is proposed for solving the LTCPP. A comparison with respect to the solution quality and execution time of our approach and another existing approach is performed.

• Chapter 3 concerns the design and implementation of the swarm intelligence family of the population-based metaheuristics for solving our problem. The common and specific search concepts of this class of metaheuristics are outlined. A Clustering Ant Colony Algorithm is proposed. Comparison is carried out to test the performance of the approach.

• Chapter 4 introduces the design and implementation of the evolutionary algorithm family of population-based metaheuristics for solving our problem. The common and specific search concepts of this class of metaheuristics are outlined. Then, a Guided Genetic Algorithm and a Multi-agent Self-adaptive Genetic Algorithm are presented. Experimental results are provided to show the efficiency and effectiveness of the two approaches.

• Chapter 5 addresses to the multi-destination daily car pooling problem. The mathematical model of MDCPP will be presented at first. Then the resolution methods, a Hybrid Ant Colony Algorithm for the MDCPP, will be outlined. At last, experimental results are examined and compared in order to evaluate the performance of the resolution method.

This thesis is concluded with a summary of our contributions and an outlook on the future work. In the appendix one, we demonstrates the platform designed for implementing and evaluating the different approaches for solving the long-term car pooling problem, as well as the platform developed for a real-world long-term car pooling

service. The structure, detailed functions and graphical interface of each platform are presented.

CHAPTER 1

An Overview of the Long-term Car Pooling Problem

CONTENTS

Abstract

In this chapter we give a general overview on the long-term car pooling problem. The mathematical model for the problem is presented. Afterwards, existing resolution approaches to the problem as well as the ones to the problems related to the long-term car pooling problem are introduced. Finally, the benchmark sets used in our experimentations are introduced

1.1 Background introduction

Nowadays, along with the increase of population and the dispersion of habitation, public transport service is often incapable of effectively servicing the areas where cost-effective transportation systems cannot be set up. As a result, more and more people use private vehicles for their daily transportation. However, the high use of private vehicles combined with increased human mobility increases the load on the environment and raises transportation issues such as congestion, parking problem and low transfer velocity.

In order to ease these issues, different innovative mobility services are emerging. Car pooling is a mobility service proposed and organized by large organizations, such as large companies, public administrations and universities. These organizations encourage their employees or students to pick up or take back colleagues or schoolmates while driving to or from a common site. The service tries to decrease the number of private vehicles travel on the road by improving the average car occupancy.

In fact, the car pooling has existed for more than 60 years. It first became prominent in the United States as a rationing tactic during World War II. It was popular in the 1970s due to the 1973 oil crisis and the 1979 energy crisis. At that time the first employee carpool programs were organized at Chrysler and 3M. However, since the 1970s carpooling has declined significantly all around the world, it peaked in the 1980 with a commute mode share of 19.7%. But since the 1990s, affected by the increasing cost of petrol and rising number of private vehicles, car pooling came back into the public eye. In the beginning of 20th century, the popularity of the Internet and mobile phones has greatly helped carpooling to expand by enabling people to find and contact car pool members more easily. With such background, the car pooling service now is experiencing the most prosperous time.

The reason why people join the car pooling system is that car pooling reduces travel costs by sharing journey expenses such as fuel, tolls and car rental between the travelers. It is also a more environmentally friendly and sustainable way to travel, as sharing journey reduces carbon emission, traffic congestion and requirement for parking space. Car pooling can also decrease driving stress since each driver has only to drive in one or two days during one week. It also creates increased social interaction between friends, neighbors and colleagues. As a matter of fact, it can enhance the sense of connectedness within the community as a small social network.

After several years of fast development, car pooling has already been considered as an important alternative transportation service throughout the world. As an effort to reduce traffic and encourage car pooling, some countries have introduced high occupancy vehicle (HOV) lanes where only vehicles with two or more passengers are allowed to drive. In some countries it is also common to find parking spaces that are reserved especially for car poolers. Many companies and local authorities have introduced car pooling schemes, often as part of wider transport programs.

Successful car pooling development has tended to be associated mainly with non-urban areas such as suburbs and more recently universities and other campuses. In the US, most of the universities have introduced the car pooling system to their students. The Seattle Smart Traveler [Meyers et al., 1999] was a funded experimental study into the proposition of a car pooling scheme at the University of Washington. It followed from earlier work in Seattle on the Bellevue Smart Traveler [Blumenthal et al., 1997]. The system collects spatial and temporal trip information using a series of WWW pages, performs a match using SQL specifications to a database engine to propose car pool members for users. Also, the Zimride system is currently being used by nearly one hundred universities and colleges, and more than 30 universities and colleges in Boise state have applied the Zipcars system to their students and employees. In France, a large number of universities also has participated the car pooling program. The largest student car pooling website, provided by RoulezMalin Company, has currently more than 50000 participants. However, these car pooling systems use only simple matching rules to propose car pool members who satisfy the time window constraints instead of considering a global optimization of the travel costs.

The main goal of this chapter is to present an overview of the long-term car pooling problem and methods from literature for its resolution. The structure of this chapter is organized as follows. Section 1.2 describes the classification and definition of the car pooling problem. The mathematical representation of the long-term car pooling problem is introduced in section 1.3. In section 1.4, we give a summary in two subsections. In the first subsection we present the existing works for solving the long-term car pooling problem, while in the second one we introduce some problems which are related to the long-term car pooling problem and their resolution methods. Section 1.5 presents the benchmarks used in our experimentations. Finally, section 1.6 gives the conclusion of this chapter.

1.2 Car pooling problem classification

According to the different procedures of using the car pooling service, we categorize car pooling problem into two different forms: Daily Car Pooling Problem (DCPP) and Long-term Car Pooling Problem (LTCPP).

In the DCPP, a number of users declare their availability for picking up or bringing back other users on one particular day. Hence, these users are considered as servers, and the other users being picked up or bringing back are considered as clients. Then the problem becomes to assign clients to servers and to identify the routes to be driven by the servers. Since in the DCPP, the servers and the clients are known in advance, the objective is to construct path starting from each server and going through as many clients as possible with respect to the car capacity and time window constraints, and to minimize the total travel cost. Based on this view, the DCPP can be considered as a

special case of the Dial-a-Ride Problem (DARP) [Healy and Moll, 1995] or Vehicle Routing Problem with Time Windows (VRPTW) [Kallehauge et al., 2005].

The DCPP model is based on daily schedule, so the participants change every day. It is a model normally used by the commercial website which organizes daily car pool service among different members. Figure 1.1 shows an example of the DCPP.

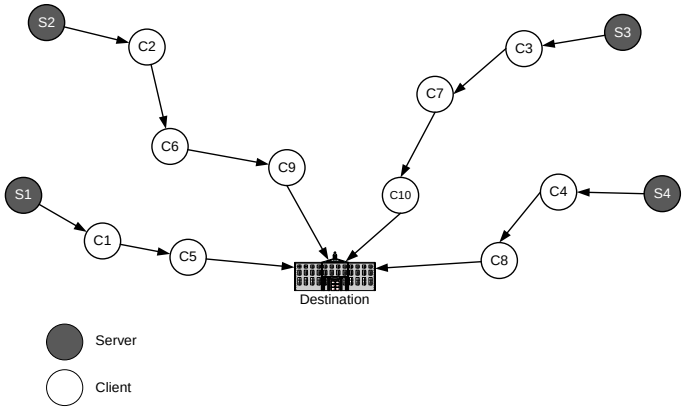


Figure 1.1: An example of the DCPP.

On the contrary, in the LTCPP, each user has to act as both a server and a client and a solution is to define car pools where each user will in turn, on different days, pick up the remaining pool members. The objective becomes to minimize the amount of vehicles used and the total distance traveled by all users, subject to car capacity and time window constraints. The LTCPP can be considered as a combination of a clustering problem and a routing problem. It requires finding the car pool members relatively close to each other and identifying the route and schedule for each member in the car pool. Figure 1.2 presents an example of LTCPP.

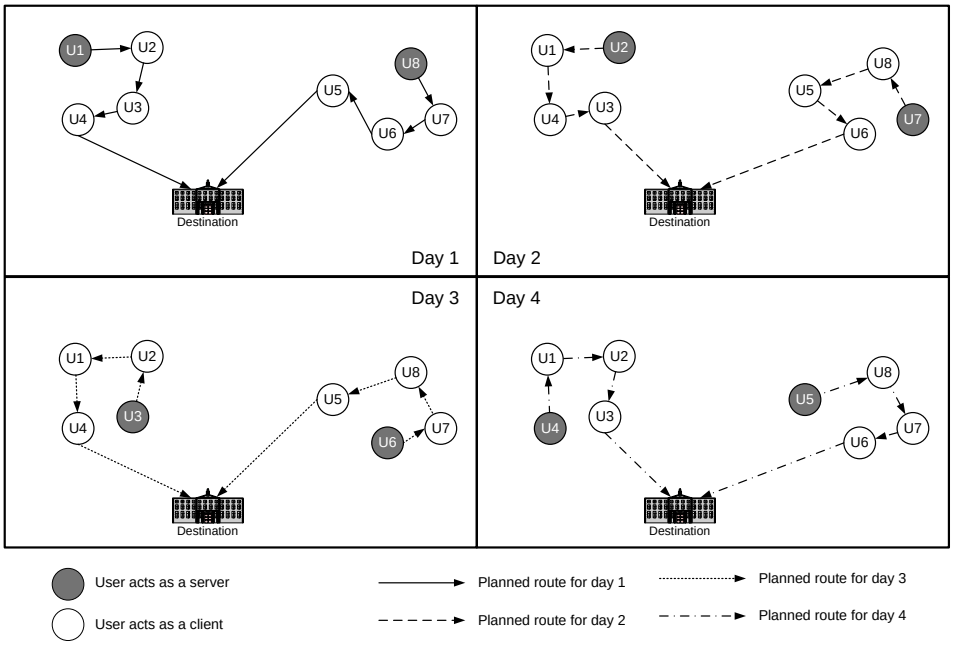


Figure 1.2: An example of the LTCPP.

The LTCPP model is a more stable car pooling model, the users in LTCPP will not change frequently in a relatively long period of time. This model is usually used by large companies, organizations and universities which provide long-term car pool service for their employees or students.

In the academic point of view, based on the similarity between DCPP and DARP and the various successful implemented approaches for DARP, solving the DCPP can simply be done by adapting the approaches from DARP with modifications.

On the other hand, we believe the LTCPP is a more valuable topic for research, since it has its own characteristics which are different from other vehicle routing problem. For instances, the problem requires clustering users into car pools based on a long-term schedule, and each car pool member has to act as a server on a different day. Thus, the server of a car pool has different departure location on different days.

In spite of its research value, the LTCPP has so far received little attention from the optimization community. Only few researches have been carried out on this problem, however, these studies are either time consuming or lacking of solution quality when dealing with large scale instances. Therefore, based on the abovementioned considerations, the LTCPP is chosen to be the focused car pooling type in our research.

1.3 Mathematical representation

In this section, we provide the definition and the mathematical formulation which are necessary for understanding the LTCPP.

1.3.1 Mathematical model

The LTCPP problem can be modeled by means of a directed graph $G = (U \cup \{0\}, A)$, where U is the set of users, and $A=\{arc(i, j) / i \in U, j \in U \cup \{0\}\}$ is the set of arcs. Each user $u \in U$ is associated with a home and node 0 represents the destination, respectively. A is a set of directed weighted arcs where each $arc(i, j) \in A$ is associated with a positive travel cost $cost_{ij}$ (which equals to the distance d_{ij} in our model) and a travel time t_{ij}. Each user enlisted in the long-term car pooling specifies: the maximal driving time T that the user is willing to accept; the earliest time e for leaving home; the latest time r for arriving at work and the capacity Q of the user's car. Note that pools are considered to be stable during a period of time and will not change frequently. This entails that the number of members in a pool will be at most equal to the capacity of the smallest vehicle among those owned by all pool members, since each member will eventually pick up all other ones.

The LTCPP is a multi-objective problem, requiring minimizing the amount of car pools and the total travel cost of all users. However, we combine these two objectives in a single objective function by using a penalty concept. The LTCPP then can be formulated as an integrated program presented as follows.

Define a pool k of users and let $|k|$ be the size of this pool. Each user of pool k, on different days, will use his/her car to pick up the remaining car pool members and then drive to the common destination. Thus each user has to find a Hamiltonian path starts at the node associated to his/her home, and then passes through all other nodes corresponding to his/her pool members' homes exactly once and ends at the common destination, shown in figure 1.3. Let $ham(i,k)$ be the above mentioned Hamiltonian path, starting from $i \in k$, connecting all $j \in k\backslash\{i\}$ and ending in 0. Suppose $|k| \leq Q_k$, where Q_k being the smallest capacity of all the cars in pool k since each car will eventually pick up all other pool members and all users' time window constraints are satisfied. The cost for a user driving to the destination directly from his/her home is denoted by $cost_{i0}$, while p_i is a penalty value incurred when the user travels alone. Then, the cost of pool k is defined in Equation (1.1). Note that the paths start from different servers and each user in a car pool must act as a server, which are the main difference between a LTCPP and a VRPTW.

$$cost(k) = \begin{cases} \dfrac{\sum_{i \in k} cost(ham(i,k))}{|k|}, & if\ |k| > 1, \\ \sum_{i \in k} cost_{i0} + p_i, & otherwise. \end{cases} \tag{1.1}$$

The total cost of a complete solution to the LTCPP is then defined to be the sum of the costs of all the pools, shown in Equation (1.2).

$$cost(K) = \sum_{k \in K} cost(k) \tag{1.2}$$

where K is the set of all car pools.

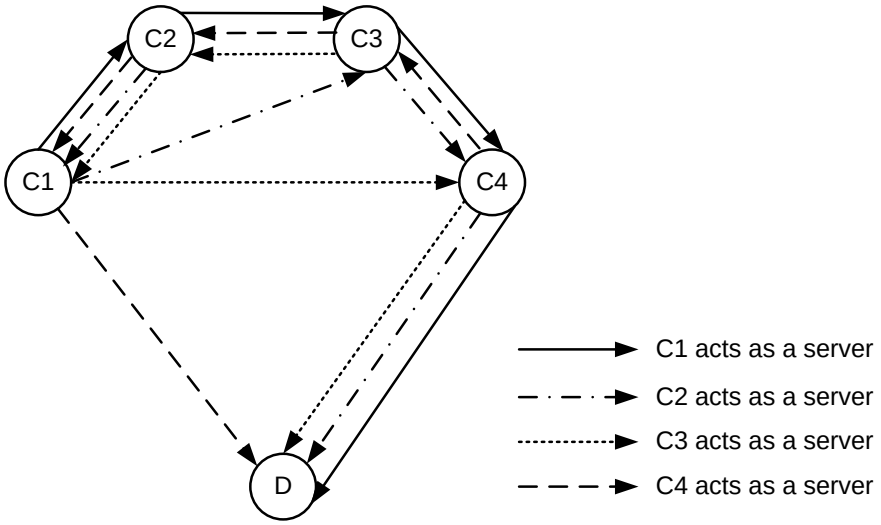


Figure 1.3: Schedules of the participants of a car pool in the LTCPP.

This view optimizes at the same time both objective functions. In our mathematical model, the penalty of a user driving alone is set to be much higher than the cost when he drives directly from his/her home to the destination, so it is always more convenient to pool users together than to leave them alone.

The LTCPP being NP is easily shown since guessing a partition is done in linear time and computing the validity of the constraints is done in polynomial time. In the paper of Varrentrapp et al. [2002], a NP-complete problem, Partition into Triangles Problem (PT), has been transformed into the LTCPP while preserving solutions, which proves the LTCPP to be NP-complete.

1.3.2 Objective function

The problem can be translated in a four index formulation considering the variables:

- x_{ij}^{hk} : Binary variable equals to 1 if *arc(i,j)* is traveled by a server *h* of a pool *k*;
- y_{ik}: Binary variable equals to 1 if user *i* is in pool *k*;
- ξ_i: Binary variable equals to 1 if user *i* is not pooled with any other user;
- S_i^h: Positive variable denoting the pick-up time of user *i* by server *h*;
- F_i^h: Positive variable denoting the arrival time of user *i* at the destination when traveling with server *h*;
- $cost_{ij}$: Positive value denoting the travel cost between users *i* and *j*;
- t_{ij}: Positive value denoting the travel time between users *i* and *j*;
- Q_k: Positive value denoting the capacity of pool *k*;
- T_h: Positive value indicating the maximal driving time when user *h* acts as a server;
- e_i: Positive value indicating the earliest time for leaving home of user *i*;
- r_i: Positive value indicating the latest time for arriving at work of user *i*;
- p_i: Positive value indicating the penalty for user *i* when he/she travels alone;
- *K*: Index set of all pools;
- *U*: Index set of all users;
- *A*: Index set of all arcs.

The objective function is shown in Equation (1.3):

$$f_{LTCPP} = \min\left(\sum_{k\in K} \frac{\sum_{h\in U}\sum_{(i,j)\in A} \text{cost}_{ij} x_{ij}^{hk}}{\sum_{i\in U} y_{ik}} + \sum_{i\in U} \rho_i \xi_i\right) \tag{1.3}$$

$$\sum_{j\in U/\{h\}} x_{ij}^{hk} = y_{ik} \qquad i,h\in U, k\in K \tag{1.4}$$

$$\sum_{j\in U} x_{ji}^{hk} = y_{ik} \qquad i,h\in U, k\in K \tag{1.5}$$

$$\sum_{j\in U} x_{ij}^{hk} = \sum_{j\in U} x_{ji}^{hk} \qquad i,h\in U, k\in K \tag{1.6}$$

$$\sum_{k\in K} y_{ik} + \xi_i = 1 \qquad i\in U \tag{1.7}$$

$$\sum_{(i,j)\in A} x_{ij}^{hk} \le Q_h \qquad h\in U, k\in K \tag{1.8}$$

$$\sum_{(i,j)\in A} x_{ij}^{hk} t_{ij} \le T_h \qquad h\in U, k\in K \tag{1.9}$$

$$S_i^h \ge e_i \qquad i,h\in U \tag{1.10}$$

$$S_j^h - S_i^h \ge t_{ij} - M\left(1 - \sum_{k\in K} x_{ij}^{hk}\right) \qquad (i,j)\in A, h\in U \tag{1.11}$$

$$F_i^h \ge S_i^h + t_{i0} - M\left(1 - \sum_{k\in K} x_{i0}^{hk}\right) \qquad i,h\in U \tag{1.12}$$

$$F_i^h \le r_i + M\left(1 - \sum_{k\in K}\sum_{j\in U} x_{ij}^{hk}\right) \qquad i,h\in U \tag{1.13}$$

$$x_{ij}^{hk} \in \{0,1\} \qquad h\in U, k\in K, (i,j)\in A \tag{1.14}$$

$$y_{ik} \in \{0,1\} \qquad i\in U, k\in K \tag{1.15}$$

$$\xi_i \in \{0,1\} \qquad i\in U \tag{1.16}$$

$$S_i^h \ge 0 \qquad i,h\in U \tag{1.17}$$

$$F_i^h \ge 0 \qquad i,h\in U \tag{1.18}$$

Equation (1.4) and (1.5) force a user *i* to be declared to be in pool *k*, if there is a path originated in *h* going from *i* to *j* or *j* to *i*; equation (1.6) is continuity constraint. Equation (1.7) forces each user to be assigned to a pool or to be penalized, while (1.8) and (1.9) are car capacity and maximal driving time constraints, respectively. Equation (1.10) and (1.11), where M is a big constant, collectively set feasible pick-up times, while (1.12) and (1.13) set minimum and maximum values of feasible arrival times, respectively. Constraints (1.14) to (1.16) are binary constraints while (1.17) and (1.18) are positivity constraints.

1.4 Related works

We present an overview of the methods designed for solving the long-term car pooling problem in this section. Since the literature of long-term car pooling problem is very limited, in order to enhance the background knowledge and obtain a comprehensive understanding of our problem, we also studied a few problems which are related to the long-term car pooling problem, and the methods designed for solving these problems.

1.4.1 Methods for solving the LTCPP

On contrary to the popularity of its related problems, only a very small amount of literature can be found for the LTCPP. Different approaches to resolve the long-term car pooling problem in literature include a Saving Functions Based Algorithm [Ferrari and Manzini, 2003], an ANTS Algorithm [Maniezzo et al., 2004], a Simulation Based Approach [Correia and Viegas, 2008] and a Multi-Matching System [Yan et al., 2011]. In this section, we categorize the solving methods into two main types: heuristics and metaheuristics.

1.4.1.1 Heuristics

During the design of an approach, heuristics are usually combined with some author defined strategies. Generally, the strategy refers to the approaches with simple policies, which facilitate or aid the heuristics by categorizing the users with restricted constraints. In the LTCPP, strategies have been normally defined to divide users into subgroups based on geographical distances or departure time differences between users. The strategies are related to specific conditions, which can only provide a general categorization or decomposition of an instance. Then, heuristics are employed to improve the performance and the solution quality of the approach.

Heuristic refers to experience-based techniques for problem solving, learning, and discovery. When an exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution. Heuristic is designed to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem. A heuristic method can accomplish its task by using search trees. However, instead of generating all possible solution branches, a heuristic is selective at each decision point, and it selects branches more likely to produce outcomes than other branches. It is intended to gain computational performance or conceptual simplicity, potentially at the cost of accuracy or precision. In heuristic, each successive iteration depends upon the step before it. Therefore, some possibilities will never be generated as they are measured to be less likely to

achieve a good solution. Heuristics essentially consists in constructive and improvement procedures. Constructive heuristics create initial solutions, i.e., set of routes, from a graph. On the contrary, as support mechanisms, improvement procedures improve previously constructed solutions by performing reassignment moves. The studies of the Saving Functions Based Algorithm, the Simulation Based approach and the Multi-Matching System fall into the categorization of heuristic.

In the Saving Functions Based Algorithm [Ferrari and Manzini, 2003], a heuristic data processing routine is designed to support efficient matching in car pool schemes. These are based on savings functions and belong to two distinct macro classes of algorithms to give two different modeling of this problem. The work is highly focused on modeling the problem instead of the solving phase. The solving methods are relatively simple by matching different users with the support of the car pool model. The approach is proven to be able to provide a large percentage in saving the travel distances in real applications. However, the approach highly depends on the distribution of the users, only the benchmarks with cluster distributed users were able to obtain good results.

The simulation-based method [Correia and Viegas, 2008] uses a divide-and-conquer approach. The heuristic that is used in the divide stage is the K-means clustering algorithm [Macqueen, 1967] which allows classifying objects based on attributes into a number of groups. The grouping is done by minimizing the sum of squares of distances between the users and the corresponding cluster centroid. The authors believe that geographic proximity does not guaranty for itself a good match between users, thus the departure and arrival time of the users was also considered as part of the distance between the users and the corresponding cluster centroid. The process starts with using the K-means clustering algorithm to divide all the users into small clusters such that each small cluster can be processed by the optimization software in an acceptable period of time. Then, all the small clusters are sent to the optimization software to search for the possible group combinations. The users that were not able to find a match in the previous iteration are set together for another iteration. The approach is tested on real-world cases. Since the approach solves the problem with the aid of optimization software, it has the ability to provide good quality solutions. However, in the real-world instances presented by the authors, the large differences between the users' time windows greatly decrease the matching rate among users.

In the study of Multi-Matching System [Yan et al., 2011], authors develop a many-to-many OD matching model, in order to perform fast grouping among the car pool users. In the model, authors define several constraints based on the geographical distance, ideal departure time difference and ideal arrival time difference between each two users, the model also requires some additional characteristics from users, such as smoking habit and gender, in order to facilitate the grouping among users. For instances, some non-smoking users require only non-smoking car pool members, some female users require only female car pool members, etc. Then, the authors develop a heuristic algorithm based on Lagrangian relaxation [Fisher, 1981] with

subgradient method [Yan, 1996] to solve the problem. The Lagrangian relaxation with subgradient method is used for the approximation of near-optimal solution. Firstly, a few constraints are relaxed to construct a Lagrangian problem, which is then solved to procure a lower bound for the optimal solution. Secondly, a Lagrangian heuristic is applied to solve for the upper bound of the optimal solution. A sub-gradient method is then utilized to revise the Lagrangian multipliers, by iterating the lower and upper bounds, until an acceptable convergence result is reached, or until the number of iterations exceeds a preset number. According to the experimental results, the model is proven to be an effective tool to group car pool members promptly. However, the computing time for large size instances usually exceeds several hours, which is considered too time-consuming for the real-world application.

1.4.1.2 Metaheuristics

Metaheuristic designates a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Metaheuristics make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. They can temporarily accept some worse solutions during the optimization procedure. Thus, it is possible to drive the search out of local optima. The rules for accepting the worse solutions are termed diversifications. With the diversification mechanism, metaheuristics can generate global optimum and are insensitive to initial solutions. However, metaheuristics do not guarantee an optimal solution is ever found. Moreover, metaheuristics contain many case-sensitive empirical parameters, which may cause some difficulties for practical implementations.

The Acronym of Approximated Non-deterministic Tree Search (ANTS) algorithm [Maniezzo et al., 2004] is the only metaheuristic can be found in the literature for solving the LTCPP.

The authors firstly define several reduction rules, and use these rules to remove from the graph representing the LTCPP problem a number of arcs which cannot belong to any feasible solution, in order to reduce the complexity of the problem. Then, to solve the problem, the authors apply the ANTS algorithm, which is an extension of the Ant Colony Optimization [Dorigo et al., 1996]. The authors specify some under defined elements of the original ACO algorithm, such as the attractiveness function to use or the initialization of the trail distribution. This turns out to be a variation of the general ACO framework that makes the resulting algorithm similar in the structure to tree search algorithms. At each stage, the algorithm has a partial solution which is expanded by branching on all possible offspring; a bound is then computed for each offspring, possibly fathoming dominated ones, and the current partial solution is selected among that associated to the surviving offspring on the basis of lower bound considerations.

Further components of the algorithm include a local search procedure, implemented as a Variable Neighborhood Search. The procedure consists of a main loop considering in turns each neighborhood. Each neighborhood is used to obtain its local optimum, and then the next neighborhood is considered. The optimization stops when no neighborhood is capable of improving the current solution.

The approach provides good solution quality, but the structures of the tree-search and the variable neighborhood search result in less efficiency when dealing with large scale instances.

1.4.2 Related problems and solving methods

1.4.2.1 Daily car pooling problem

The daily car pooling problem is structurally different from the long-term car pooling problem. Unlike the LTCPP which each user has to act both as a server and a client, the roles of a server and a client are fixed and known beforehand at the beginning. Therefore, the DCPP focuses on constructing the routes start with each existing server. The methods of assigning clients to servers in the DCPP can be interesting for the LTCPP. Thus, the approaches for solving the DCPP become valuable in our research.

The DCPP is NP-hard since in a special case it contains the Vehicle Routing Problem with unit client demands, which is known to be NP-hard in the strong sense. Some authors [Kothari, 2004; Vargas et al., 2008; Maurizio et al. 2011] define a few simple matching rules to build car pools in order to obtain fast matching speed; however, a good solution quality cannot be guaranteed.

For instance, in the work of Kothari [2004], a multi-agent car pooling system called Genghis system is developed. The system is designed through the Gaia methodology and implemented on a FIPA-compliant Jade platform. In the car pool building phase, two primitive pool types are built, and the authors define a few fundamental constraints for choosing reasonable matches between clients and servers. An algorithm is developed to generate route proposals for the users. The algorithm picks a client only once and continues matching the client to proposed servers until all the proposed servers are examined. The proposed server is selected based on the distance between the client and the direct route from the server to the server's destination. For each client, the algorithm proposes a maximum 5 available drivers, then for each proposed driver, the algorithm matches him to the client in a hierarchy of 4 Levels. Each level consists in a matching constraint, such as user preference, distance constraint, time window constraint and cost constraint. The client may be rejected at any level of the matching. Each match will be evaluated and rated, and the matching which has the highest rating will be selected. Then the algorithm will select another client and continue the matching process until all the clients are matched with servers

or all servers' capacities are reached. The Genghis system is designed for solving small size instances in an efficient way, but minimizing the total travel cost is not taken into consideration in this system.

To obtain a good solution quality, other authors use heuristics and exact algorithms to solve the DCPP. The interesting studies include a distributed geographic information system [Calvo et al., 2004] and an exact algorithm based on Lagrangian Column Generation [Baldacci et al., 2004].

In the distributed geographic information system, the authors present an integrated system for the organization of a car pooling service. The core of the system is an optimization module which solves heuristically the specific routing problem. The procedure includes a construction phase and a local optimization phase. The algorithm starts with no employee routed and an idle fleet of vehicles. A number of new routes equal to the number of servers are initialized as direct paths from each server to the destination, then, as long as possible, single clients are inserted into existing routes with a greedy algorithm. Then, from the initial solution produced by the procedure described above, better ones are obtained by means of a local search algorithm.

In the study of the exact algorithm based on Lagrangian Column Generation, authors propose an exact method for the car pooling problem. This method is based on two integer programming formulations of the DCPP. The first formulation is a commodity flow formulation using three-index variables, while the second formulation models the DCPP as a set-partitioning problem whose variables correspond to feasible paths or to clients to be left unserved. A valid lower bound on the optimal DCPP cost is computed as the cost of a feasible dual solution of the LP relaxation of the set-partitioning problem; the solution is obtained by combining three different relaxations of the two formulations. The dual solution and a valid upper bound obtained by a heuristic algorithm based on the bounding procedure are then used to eliminate feasible paths that cannot belong to any optimal solution; thus the resulting reduced set-partitioning problem can be solved by a branch-and-bound algorithm. The main contributions of this research are the new bounding procedures for computing a feasible dual solution of the set-partitioning formulation and the method for generating a reduced set-partitioning problem that is used to find an optimal solution.

1.4.2.2 Dial-a-ride problem

The dial-a-ride problem (DARP) involves designing vehicle routes and schedules to satisfy a set of travel requests. The vehicle fleet departs from one or several depots. A travel request consists of picking up a certain client at a predetermined pickup location during a specified departure time interval and transporting the client to a predetermined drop off location to be reached within a specified arrival time interval. The departure and arrival time windows are based on desired pickup or delivery time requests specified by the client. The aim is to design a set of minimum cost vehicle routes capable of serving as many requests as possible, under a set of constraints.

Based on the modeling point of view, the DARP is a generalization of the capacitated pickup and delivery problem with time windows (PDPTW), which was first studied by Wilson et al. [1971]. The DARP is similar to DCPP in many perspectives. Both of them have to define routes to pick up and drop off clients within predefined time windows. The main characteristics make DARP different from DCPP are the departure and arrival points of the vehicles. In DARP, all vehicles depart from one or several fixed depots and return to these depots after the service, whereas the servers in DCPP depart from their own homes which are random distributed and end the trip at a common destination of the passengers in the vehicles. In the literature, it is common to consider a DCPP as a special type of DARP, which makes the studies on DARP become interesting to our LTCPP research.

Unlike DCPP and LTCPP, extensive studies were carried out on DARP research. DARP services may operate in a static or a dynamic mode. Since in practice the transportation requests are usually known in advance, most of the researches have focused on static DARP. The DARP has been proven to be NP-hard [Healy and Moll, 1995], but efforts were still made to solve the problem with exact algorithms. Early approaches [Psaraftis, 1980] dedicate to solve single-vehicle problems using pure dynamic programming (DP) method. However, due to the complexity of the problem, the amount of vehicles is limited to one per problem. Then Desrosiers et al. [1986] introduced the concept of dominance to reduce intermediate states. This technique greatly improves the speed of the DP process if the problem is subjected to strong constraints. Based on this approach, some multi-vehicle problems can be exactly solved by the combination of the column generation method and the branch and bound process [Dumas et al., 1989]. But the exact approach still has a strong limitation on the size of instances it can solve.

Because of the complexity of the DARP and the large size instances in real application, the most popular approaches are still heuristics. Sexton and Bodin [1985a; 1985b] developed an insertion heuristic algorithm to solve the problem. The objective of the algorithm is to minimize a user's inconvenience function, which is defined based on the weighted sum of two values. The first value measures the difference between the actual travel time and the direct travel time of a server. The second value calculates the difference between the willing drop off time and the actual drop off time of a passenger. Jaw et al. [1986] proposed a heuristic approach which selects users in the order of earliest feasible pickup time and gradually inserts them into vehicle routes in order to yield the least possible increase of the objective function. Other sequential insertions are also commonly used. For instance, in the work of Cordeau et al. [2001], the insertion is performed according to the nearest distance or the minimum cost. Toth and Vigo [1997] have proposed a heuristic method. The method firstly assigns requests to routes by means of a parallel insertion procedure, and then performs intra-route and inter-route exchanges. The tests show significant improvement, the exchange phase is very useful in optimize the solution obtained in the parallel insertion phase.

Another well-known method, which is cluster first and route second [Bodin and Sexton, 1986] is also applied in solving the DARP. In the method, the geographically close clients are clustered together before applying the routing algorithm to each cluster. In the first step, a large set of clusters is constructed and a set partitioning problem is then solved to select a subset of clusters serving each user exactly once. Then in the second step, feasible routes are enumerated by combining clusters, and a second set partitioning problem is solved to select the best set of routes covering each user exactly once. Since the time windows could be different between the clients geographically close, the method of mini-clusters is developed [Ioachim et al., 1995]. The mini-clusters consider grouping the geographically close users who have the similar time windows.

Calvo and Colorni [2002] have proposed a heuristic for a particular version of the DARP where the number of available vehicles is fixed as well as the time windows for picking up and dropping off passengers. The algorithm first attempts to service as many users as possible and then minimizes user inconvenience expressed as the sum of waiting time and excess travel time. The heuristic firstly constructs a set of routes and a number of sub-tours by solving an assignment problem. A routing phase is then performed to insert the sub-tours in the routes and to re-sequence the clients within the routes.

Tabu search is also applied to the DARP by Cordeau and Laporte [2003]. The passengers have to specify a time window on the arrival time of their outbound trip and on the departure time of their inbound trip, and a maximum ride time is also associated with each passenger. Capacity and maximum route length constraints are imposed on the vehicles. The search algorithm iteratively removes a transportation request and reinserts it into another route. Infeasible solutions are allowed during the search by using a penalized objective function. Also, the minimum duration schedule associated with each candidate solution is computed. According to the experimental results, the approach can provide good solution for large size instances.

1.4.2.3 Vehicle routing problem with time windows

Generally, a vehicle routing problem with time windows (VRPTW) is to use a set of vehicles to serve a group of clients. The objective of a VRPTW involves delivering goods from a depot to a set of geographically scattered clients. The routes must be designed in such a way that each point is visited only once by exactly one vehicle within a given time interval; all routes start and end at the depot. The vehicles have limited carrying capacity and the total demands of all points on one particular route must not exceed the capacity of the vehicle. The VRPTW has multiple objectives which are to minimize not only the number of vehicles required, but also the total travel time or total travel distance incurred by the fleet of vehicles.

The VRPTW is a well-known problem widely studied by the optimization community and has a huge number of publications. The methods developed for VRPTW

can be a very important asset to any transportation related problems. Because of the lack of literature in LTCPP, we studied the metaheuristics for solving VRPTW, in order to gain some inspiration for developing our approach for solving the LTCPP.

Tabu search (TS) is a popular approach for solving VRPTW. It is a local search metaheuristic introduced by Glover [1986]. The initial solution of TS is normally created with some simple insertion heuristic. The most common one is Solomon's insertion heuristic [Solomon, 1987]. Other insertion heuristics can also be easily found in literature. De Backer and Furnon [1997] use a savings heuristic; Tan et al. [2000] perform a modified Solomon's insertion heuristic, and Cordeau et al. [2001] use a modified sweep heuristic. Lau et al. [2003] introduce the concept of a holding list, which is a data structure containing the unserved clients. In the beginning all clients are stored in the holding list, and simple relocate and exchange operators are then used to move clients up and down from the holding list.

After creating an initial solution, the usual next step is to improve it using local search with one or more neighborhood structures. Most of the neighborhoods used are well known, such as 2-opt, Or-opt, relocate, and exchange. In order to reduce the complexity of the search, the authors usually propose their own special strategies for limiting the size of the neighborhoods. For instance, Garcia et al. [1994] only allow moves involving arcs that are close in distance. Taillard et al. [1997] decompose solutions into a collection of disjoint subsets of routes by using the polar angle associated with the center of gravity of each route. Tabu search is then applied to each subset separately. A complete solution is reconstructed by merging the new routes found by tabu search. To maintain the diversity of their search, some authors allow infeasibilities during the search, and the violations of the constraints are punished by using a penalty concept in the objective function. For instance, Lau et al. [2003] allow violation of the vehicle capacity and time windows constraints, but penalize the violations of constraints in the cost function.

The genetic algorithm (GA) is the most favored approach on VRPTW. GA evolves a population of individuals encoded as chromosomes by creating new generations of offspring through an iterative process until some convergence criteria are met. The best chromosome generated is then decoded, providing the corresponding solution. Although theoretical results that characterize the behavior of the GA have been obtained for bit-string chromosomes, not all problems can be easily represented in this way. For the vehicle routing problem, an integer representation is more often selected, since it is more convenient in corresponding to the problem. Therefore, in most applications to VRPTW, the genetic operators are applied directly to solutions, represented as integer strings, thus avoiding encoding and decoding operations. The creation of a new generation of individuals involves four major steps: initialization, selection, recombination, and mutation.

The initial population is typically created either randomly or using modifications of well-known construction heuristics. In the work of Thangiah et al. [1995], the clients are randomly clustered into separate groups and then a cheapest insertion heuris-

tic is used to route clients within each group. Berger et al. [2003] modify a randomly generated initial population with exchanges and a re-initialization procedure based to create a population of solutions with the number of vehicles equal to the lowest found.

The most typical selection method for selecting a pair of individuals for recombination or mutation is the well-known roulette wheel selection mechanism. In this stochastic mechanism, the probability of selecting an individual is proportional to its fitness value. Tan et al. [2001a] and Jung and Moon [2002] propose the tournament selection. The basic idea is to perform the roulette wheel selection twice, and to select the better out of the two individuals identified by the roulette wheel selection mechanism. The tournament selection becomes more and more popular recently since it has less stochastic noise, and has a constant selection pressure.

The recombination, also called crossover, is the most essential part of a genetic algorithm. The traditional two-point crossover, which exchanges a randomly selected portion of the bit string between the chromosomes, is commonly used, while Tan et al. [2001b] use the well-known PMX and one-point crossovers. The basic idea in PMX crossover is to choose two cut points at random and, based on these cut points, to perform a series of swapping operations in the second parent. The one-point crossover switches two sets of clients to be serviced by two different routes. In the context of VRPTW, many authors have proposed specialized heuristic crossover procedures, instead of traditional operators. Potvin and Bengio [1996] propose a sequence-based and a route-based crossover. The sequence-based crossover first selects a link randomly from each parent solution. Then, the clients that are serviced before the break-point on the route of one parent solution are linked to the clients that are serviced after the break-point on the route of the other parent solution. Finally, the new route replaces the old one in the first parent solution. The route-based crossover replaces one route of the second parent solution by a route of the first parent solution. In Berger et al. [2003], a removal procedure is first carried out to remove some clients from the solution. Then, an insertion procedure is locally applied to reconstruct the partial solution. Wee Kit et al. [2001] tries to change the order of the clients in the first parent by trying to create consecutive pairs of clients according to the second parent. The second crossover operator tries to copy common characteristics of parent solutions to offspring by modifying the seed selection procedure and cost function of an insertion heuristic.

Another important strategy of the genetic algorithm is mutation. Mester [2002] uses a multi-parametric mutation that consists in removing a set of clients from a solution randomly, based on the distance to the depot or by selecting one client from each route. Then, a cheapest insertion heuristic is used to reschedule the removed clients. In Gehring and Homberger [2001] mutation is also used to reduce the number of routes by performing one or several subsequent relocate moves. Berger et al. [2003] present several mutation operators including the LNS, exchange of clients served too late in the current solution, and elimination of the shortest route.

Another metaheuristic used in solving VRPTW is simulated annealing. Tan et al. [2000] develop a fast simulated annealing method based on two-interchanges with best-accept strategy and a monotonously decreasing cooling scheme. After the final temperature is reached, special temperature resets based on the initial temperature and the temperature that produced the current best solution are used to restart the procedure. Li et al. [2003] propose a tabu-embedded simulated annealing metaheuristic. Initial solutions are created by the insertion and extended sweep heuristics of Solomon [1987]. Three neighborhood operators based on shifting and exchanging clients between and within routes are combined with a simulated annealing procedure that is forced to restart from the current best solution several times. Solomon's insertion procedure is used to reduce the number of routes and to intensify the search by reordering routes and trying to insert clients into other routes. Finally, the search is diversified by performing some random shifts and exchanges of clients.

The ant colony optimization (ACO) is also applied to solve the VRPTW. Gambardella et al. [1999] use an ant colony optimization approach with a hierarchy of two cooperative ant colonies. The first colony is used to minimize the number of vehicles, while the second colony minimizes the total traveled distance. The two colonies cooperate through updating the pheromone with the best found solution. When the new best solution contains fewer vehicles, both colonies are reinitialized with the reduced number of vehicles.

Bräysy [2003] presents a new four-phase deterministic metaheuristic algorithm based on a modification of the variable neighborhood search (VNS). In the first phase, an initial solution is created using a construction heuristic based on the ideas of the works of Solomon [1987] and Russell [1995]. Routes are built one at a time in a sequential order. Then, after a number of clients have been inserted into the route, the route is reordered using Or-opt exchanges. Afterward, another operator is used to minimize the number of routes. In the third phase, the created solutions are improved in terms of distance using VNS oscillating between four new improvement procedures based on modifications to CROSS-exchanges of Taillard et al. [1997] and cheapest insertion heuristics. In the fourth phase, the objective function used by the local search operators is modified to also consider waiting time to escape from local optima.

1.5 Benchmarks

As no benchmark particularly designed for the LTCPP has been made public, we developed our own benchmark to aid our experiments. The benchmark used in our experiments includes three sets of structurally different problem instances. Two of them are transformed from the benchmark of a similar problem and the other one is obtained based on the real-world case.

The first two sets of instances were originally derived from the Pickup and Delivery Problems with Time Windows (PDPTW) instances by Li and Lim [2003]. We added and modified a few values in order to transfer them into LTCPP instances. Both of the two sets are composed of 9 instances with users from 100 to 400. The users in the first set are clustered distributed, so the set is named with C. The second set has the users allocated randomly, therefore is named with R.

For all the instances in these two sets, the depot in the original PDPTW of 100 clients is considered as the destination, while the coordinates the users are kept from the original benchmark. The values of maximum travel time T_k, the penalty p_i, the earliest departure and latest arrival time e_i and l_i of each user, were generated according to the research on the real-world applications. The cost d_{ij} was computed as an integer value equal to ed_{ij}, where ed_{ij} is the Euclidean distance between user i and j. Travel times t_{ij} were assumed to be equal to the distances divided by *50 km/h* (average travel speed). For each user, the car capacity Q_k was set to *4*, and the maximum ride time T_k was defined to be *1.5* times of the direct travel time from the user's home to the destination. The penalty p_i of each user was computed as two times of travel cost from user's home directly to the destination. The latest arrival times l_i were uniform randomly selected in the interval from *8:30 am* to *9:00 am*, and earliest departure time of user i was computed as $e_i = l_i - max\ (t_{i0} + 0{:}30,\ 2t_{i0})$, where t_{i0} is the direct travel time from the user i's home to the destination.

The last set of instances is obtained by real-world case. The data is collected from the car pooling program participants of the Artois University by using the car pooling platform presented in the appendix. The university is the destination in these instances. The participants defined their earliest departure time from their homes, latest arrival time at the university, their car capacity and the maximal travel time they were willing to take. However, the car capacity cannot be set less than *2*, and the maximum driving time cannot be less than *1.2* times of the direct travel time from the user's home to the university. The distances and travel time between each two users and between the user and the destination are obtained by Google Map API, which provides very accurate values for the instances. The collected data includes 565 participants. The data is first transformed into an instance with 565 users directly as well as two 400 users instances by selecting randomly 400 users from the data. Then three instances with 200 users are generated by each time randomly selecting 200 users from the data. Moreover, three instances with 100 users are also built by randomly selecting 100 users from the data. Thus, the last set contains 9 instances based on the real-world data.

The benchmarks can be found at http://www.lgi2a.univ-artois.fr/agora/module_fichier. The benchmarks we defined in this thesis are considered as very large instances for a LTCPP, since in the literature the sizes of instances are between 25 and 200. Moreover, our benchmarks contain users traveling in a similar time range and a relatively small area; this greatly increases the amount of possible solutions which further increases the difficulty of the instances. Thus, our benchmarks can be considered as hard instances for the LTCPP.

1.6 Conclusion

An overview of different aspects of long-term car pooling problem has been presented in this chapter. This problem is interesting in research as well as in real world application because of its unique characteristics and the challenge it provides.

The state-of-the-art presented in this chapter covers the description of the problem, the mathematical representation and the existing solving methods. The resolution methods for Daily Car Pooling Problem, Dial A Ride Problem and Vehicle Routing Problem with Time Windows are also introduced in detail in order to supplement the literature of LTCPP. The benchmark sets which are used for our algorithm experimentations are presented in detail.

According to all the literature we studied, we can conclude that a well-designed approach should not only focus on a given class of methods, but also has to take into account different features and mechanisms that have been employed to aid or support these methods.

Next chapters will handle these open issues, we will expose our contribution in the field of long-term car pooling problem. The proposed approaches are designed by taking advantage of the best existing approaches for solving all related problems.

CHAPTER 2

Trajectory-based Metaheuristics for the Long-term Car Pooling Problem

CONTENTS

Abstract

In this chapter we propose to solve the LTCPP with Variable Neighborhood Search approach which is a well-known member of the trajectory-based metaheuristic family. We believe that the characteristic of changing the neighborhood structure offers a powerful mechanism in finding good solutions. Experimental results are presented to show the performance of the approach.

2.1 Introduction

In our first attempt to solve the long-term car pooling problem, we dedicated to developing and applying a simple method, because a simple method usually brings facilitation in implementation and robust in operation. Therefore, the trajectory-based metaheuristics are taken into our consideration. The trajectory-based metaheuristics typically normally process one solution at a time. They can trace out a path in the search space as the iterations continue.

One of the most recent approaches in the field of trajectory-based metaheuristics is Variable Neighborhood Search (VNS) [Hansen and Mladenović, 1997]. The metaheuristic is inspired by the fact that a local optimum related to a specific move type can often be improved by using another move type. To exploit this fact, the VNS defines different move types and change the move type used once a local optimum has been obtained. The search space covered by a specific move type is called a neighborhood, so the VNS adaptively changes the neighborhood and obtains different local optima. If we consider searching inside one neighborhood is an intensification search process, then the process of changing neighborhoods corresponds to a diversification search process. Different neighborhoods cover different search spaces, and the properties of one neighborhood are in general different from those of other neighborhoods, therefore the search strategies usually are different for each of them. When the neighborhoods are well designed, a solution that is locally optimal to one neighborhood is usually not locally optimal to another neighborhood. So, the global optima can be found in the local optima of the neighborhoods. This characteristic provides to VNS a serious ability and reactivity to track the shifting optimum in optimization problems. Therefore, in this chapter a Variable Neighborhood Search approach is proposed for solving the LTCPP. We believe that the characteristic of changing the neighborhood structure offers a powerful mechanism in finding good solutions.

The structure of this chapter is organized as follows: the fundamental explanation of trajectory-based metaheuristics is introduced in the Section 2.2. Section 2.3 presents our VNS-LTCPP in an incremental manner, with the representation, the definition of the neighborhood structure, the design of the initial solution, the evaluation function, and the dedicated algorithm. In Section 2.4, we discuss our experimental results, and provide an experimental result analysis. Finally, in section 2.5, we conclude with a summary of the main contributions reported in this chapter.

2.2 Trajectory-based Metaheuristics

As abovementioned, trajectory-based algorithm typically operates on one solution at a time, which will trace out a path in search space as the iterations continue. Paths are

performed by iterative procedures that allow moving from one solution to another one in the solution space. The walks start from a solution randomly generated or obtained from another optimization algorithm, called initial solution. In each iteration, the current solution is replaced by another one selected from the set of its neighboring candidates. A better move or solution is always accepted, while a not-so-good move can be accepted with certain probability, shown in figure 2.1. The steps or moves trace a trajectory in the search space, with a non-zero probability that this trajectory can reach the global optimum. The search process is stopped when a given condition is satisfied, such as a maximum number of generations, finding a solution with a target quality, or no improvements for a given time, etc. This kind of metaheuristics performs the moves in the neighborhood of the current solution, so they have a perturbative nature.

***Algorithm 2.1**: Trajectory-based Metaheuristic.*

Generate (s_0); / Generate initial solution */*
t = 0; / Number of iterations */*
s_t = s_0;

While** not Termination Criterion (s_t) **do
    s_t' = GenerateMove (s_t); / Exploration of the neighborhood */*
    if** AcceptMove(s_t') **then
    s_t = ApplyMove (s_t'); / Replace incumbent with the new obtained solution */*
    t = t+1;
End while

Algorithm 2.1 illustrates the structure of a trajectory-based metaheuristic. It iteratively applies the generation and replacement procedures from the current single solution. In the generation phase, a candidate solution s_t' is generated from the current incumbent s_t. The candidate solution is generally obtained by local transformations of the solution. In the replacement phase, an evaluation is performed for the candidate solution s_t' to replace the current incumbent s_t. If the candidate solution s_t' is proven to be better than s_t, then s_t' will be accepted and selected to become the new incumbent by replacing s_t. This process iterates until a given stopping criteria is met. In this algorithm, the generation and the replacement phases has no memory mechanism, the procedures are based only on the current incumbent. However, in some of the trajectory-based metaheuristics, some experiences of the former searches stored in a memory can be used in the generation of the candidate list of solutions and the selection of the new solution.

The main concepts for the trajectory-based metaheuristics, which are the definition of the neighborhood structure and the determination of the initial solution, will be introduced in the next section.

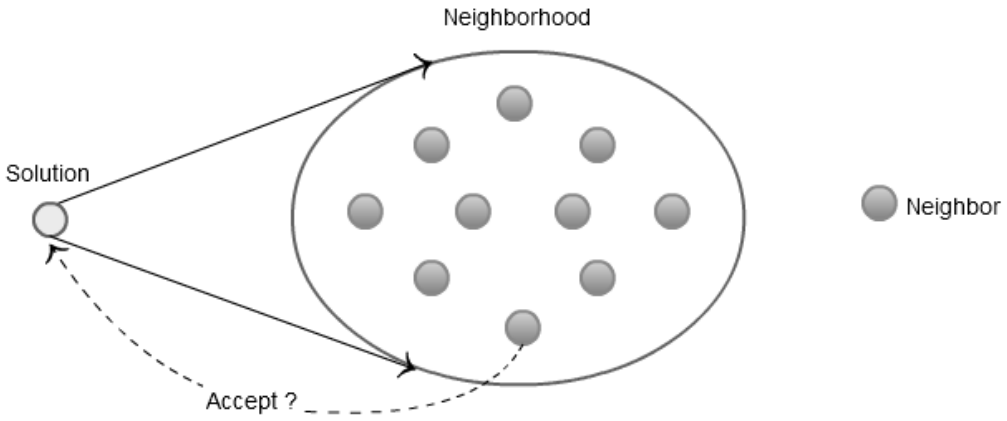


Figure 2.1: Main principles of the trajectory-based metaheuristic.

2.2.1 Neighborhood

The most basic and essential part of a trajectory-based metaheuristic is the definition of the neighborhoods. The structure of the neighborhood plays a crucial role in the performance of a trajectory-based metaheuristic. If the neighborhood structure is not adequate to the problem, the trajectory-based metaheuristic will either fail to solve the problem or provide low quality solutions.

Definition 2.2.1
If X is a search space and s is a solution in X, and all the solutions in X are connected with a defined mapping rule, then a neighborhood of s is a set N(s) $\subset$ X containing the solutions where s can move to without leaving the search space. A solution $s_n \in N(s)$ is a neighbor of s[Hansen and Mladenovic, 1997].

A neighbor is constructed by applying of a move operator which performs a small perturbation to the current solution. The main characteristic of a neighborhood is locality. Locality is the effect on the solution when performing the move in the representation of the solution. The neighborhood is considered to have a strong locality, if when small changes are made in the representation, the solution is affected slightly. Hence, a trajectory-based metaheuristic will perform a meaningful search in the search space of the problem. Weak locality is characterized by a large effect on the solution when a small change is made in the representation of the solution. In the extreme case of weak locality, the search will converge toward a random search in the

search space. The structure of the neighborhood depends on the target optimization problem. It has been first defined in continuous optimization.

Definition 2.2.2

In a discrete optimization process, the neighborhood N(s) of a solution s is represented by the set {s_n / d(s_n, s) ≤ ε}, where d represents a given distance that is related to the move operator[Hansen and Mladenovic, 1997].

The definition of the neighborhoods strongly corresponds to the representation associated with the problem to solve. Normally in vehicle routing related problems, the neighborhoods are designed with traditional encodings; the users are represented in a sequence of numbers, called permutation. For permutation-based representations, a usual neighborhood is based on the swap operator that consists in exchanging the location of two elements s_i and s_j of the permutation. For a permutation of size n, the size of this neighborhood is $n(n-1)/2$. This representation may also be applied to other linear mathematical models. Figure 2.2 shows the neighborhood associated with a combinatorial optimization problem using a permutation encoding. In the figure, the neighbors of the solution (2, 1, 3) are: (2, 3, 1), (1, 2, 3), and (3, 1, 2).

The distance between two elements is based on the swap operator. Once the concept of neighborhood has been defined, the local optimality property of a solution may be given.

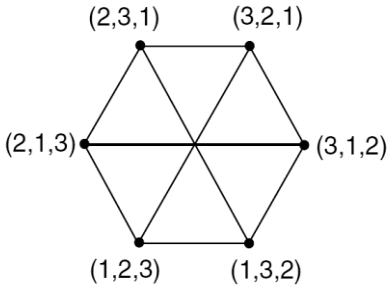


Figure 2.2: An example of neighborhood for a permutation problem.

Definition 2.2.3

For a given neighborhood N(s), a solution s ∈ N(s) is a local optimum if the solution quality is better than or equal to all its neighbors' solution qualities, see figure 2.3. That is, Q(s) is better than Q(s_n) for all s_n ∈ N (s). Note that for the same optimization problem, a local optimum for a neighborhood N_i may not be a local optimum for another neighborhood N_j [Hansen and Mladenovic, 1997].

Definition 2.2.4

For a search space X, a solution s ∈ X is a global optimum if the solution quality is better than or equal to all others solution quality in the search space. That is, Q(s) is

better than $Q(s_n)$ for all $s_n \in X$. Note that there may be many global optimal solutions in a search space [Hansen and Mladenovic, 1997].

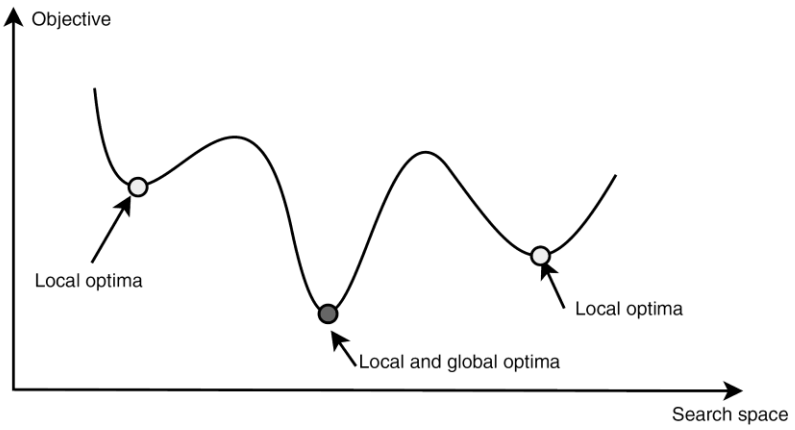


Figure 2.3: Local optimum and global optimum in a search space.

2.2.2 Initial Solution

To construct an initial solution, two strategies are normally used: a random procedure and a greedy approach. There is a lot to discuss between the selection of a random initial solution and a greedy initial solution, since a random initial solution is superior in terms of the computational time but is lack of solution quality, while a greedy initial solution does vice versa. The best answer to this trade-off will depend mainly on the efficiency and effectiveness of the random and greedy algorithms at hand, and the trajectory-based metaheuristic properties. For instance, the larger is the neighborhood, the less is the sensitivity of the initial solution to the performance of the trajectory-based metaheuristics. Generating a random initial solution is a quick operation, but the metaheuristic may take much larger number of iterations to converge. To speed up the search, a greedy heuristic may be used. In fact, in most of the cases, greedy algorithms have a reduced polynomial-time complexity. Using greedy heuristics often leads to better solution quality. Thus, the trajectory-based metaheuristic will require less iterations to converge toward a local optimum. Some approximation greedy algorithms may also be used to obtain a bound guarantee for the final solution. However, it does not mean that using better solutions as initial solutions will always lead to better local optima.

2.2.3 Some Trajectory-based Metaheuristics

Popular examples of trajectory-based metaheuristics are Hill Climbing (Russell and Norvig, 2003), Greedy Randomized Adaptive Search (Feo and Resende, 1989), Simu-

lated Annealing (Kirkpatrick 1983, Cerny 1985), Tabu Search (Glover, 1990), and Variable Neighborhood Search (Hansen and Mladenovic, 1999).

- Hill Climbing (HC) is a trajectory-based metaheuristic that normally starts with a random initial solution to a problem. Then in each iteration, HC will adjust a single element in the current solution and determine whether the change improves the value of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

- The Greedy Randomized Adaptive Search Procedure (GRASP) typically consists of iterations made up from successive constructions of a greedy randomized solution and subsequent iterative improvements of it through a local search. The greedy randomized solutions are generated by adding elements to the problem's solution set from a list of elements ranked by a greedy function according to the quality of the solution they will achieve. To obtain variability in the candidate set of greedy solutions, well-ranked candidate elements are often placed in a restricted candidate list, and chosen at random when building up the solution.

- Simulated Annealing (SA) is a stochastic search method in which at each step, the current solution is replaced by another one that improves the objective function. The replacement is normally randomly selected from the neighborhood. SA uses a control parameter, called temperature, to determine the probability of accepting non-improving solutions. The objective is to escape from local optima, and so to delay the convergence. The temperature is gradually decreased according to a cooling schedule such that few non-improving solutions are accepted at the end of the search.

- Tabu Search (TS) explores the search space by managing a memory of solutions or moves recently applied, called the tabu list. When a local optimum is reached, the search carries on by selecting a candidate worse than the current solution. To avoid the previous solution to be chosen again, and so to avoid cycles, TS discards the neighboring candidates that have been previously applied.

- Variable Neighborhood Search (VNS) is a trajectory-based metaheuristic which explores successively a set of pre-defined neighborhoods to provide a better solution. It uses the descent method to get the local minimum of one neighborhood. Then, it explores either at random or systematically other neighborhoods. At each step, a solution is shaked from the current neighborhood. Then the current solution is replaced by a new one if and only if a better solution has been found. The exploration is thus re-started from that solution in the first neighborhood. If no better solution is found the algorithm moves to the next neighborhood, randomly generates a new solution and attempts to improve it.

2.3 Variable Neighborhood Search for the LTCPP

In this section, we present our VNS-LTCPP approach designed for solving the long-term car pooling problem. The adaptation of the different components for the long-term car pooling problem is described and examined.

2.3.1 Variable Neighborhood Search

Variable Neighborhood Search (VNS) has been recently proposed by P. Hansen and N. Mladenovic (1999). Contrary to most other trajectory-based metaheuristics, VNS follows more than one trajectory. It explores increasingly distant neighborhood of the current incumbent solution, and it jumps from there to a new one if and only if an improvement was made. VNS exploits the fact that using various neighborhoods in local search may generate different local optima, and that provides the possibility to find the global optimum among the local optima of the given neighborhoods, since different neighborhoods search different areas of the search space. Figure 2.4 shows the mechanism of a VNS with two neighborhoods. The first local optimum is obtained according to the neighborhood 1, while the second local optimum is obtained from neighborhood 2 based on the previous local optimum. In this way the favorable characteristics of the incumbent solution will be kept and used to obtain promising neighborhood solutions.

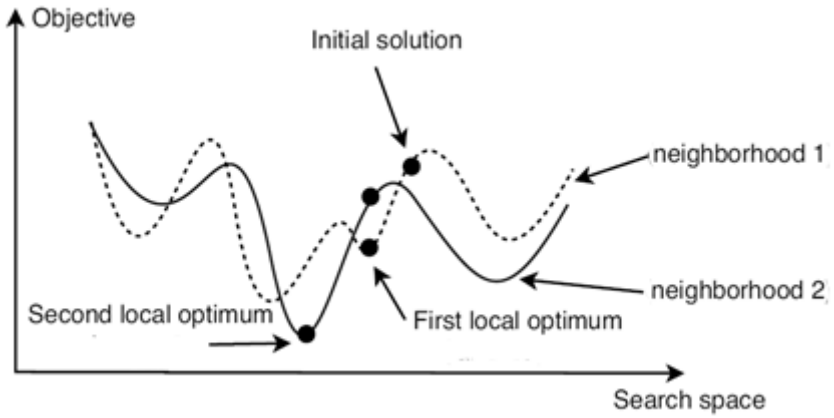


Figure 2.4: Variable neighborhood search using two neighborhoods.

In VNS, a finite set of pre-selected neighborhood structures N_k *(k = 1, ..., n)* are defined. Then, each iteration of the algorithm consists in three steps: shaking, local search and move. A solution is firstly shaken from the neighborhood N_k of the current solution. For instance, a solution *s'* is randomly selected in the current neighborhood N_k of current incumbent *s*. Then, a local search operator is applied to the solution *s'* to

generate a local optimal solution *s''*. The current incumbent *s* is replaced by the new solution *s''* if and only if *s''* is a better solution than *s*, that is the solution quality *Q(s'')* is better than the solution quality *Q(s)*. Afterward, the search procedure is re-started with the new incumbent *s*. If the quality of the new solution *Q(s'')* is worse than the solution quality *Q(s)*, the algorithm moves to the next neighborhood. The procedure continues until some stopping condition is met, for instance, maximum number of iterations, maximum CPU time allowed, or maximum number of iterations between two improvements. Algorithm 2.2 presents the template of the basic VNS algorithm.

***Algorithm 2.2**: Basic Variable Neighborhood Search Algorithm*

Define the set of neighborhood N_k (k = 1, … , n);
Generate (s_0); / Generate initial solution */*
s = s_0;

Repeat
  k = 1;

  Repeat
    Generate solution s' at random from N_k(s); / Shaking */*
    s'' = LocalSearch(s'); / Apply local search to obtain local optimum s'' */*

    If** Q(s'') < Q(s) **then
      s = s''; / Move */*
      k = 1; / Start the next search in the first neighborhood N_1(s) of solution s */*

    Else
      k = k + 1;

    End if

  ***Until** k = n*

***Until** stopping criteria is met*

2.3.2 LTCPP Solution Representation

The aim to design a representation for the solution is to build a suitable mapping between our problem and the solution generated by the algorithm. Although both direct and indirect coding have been proven to be applicable for the representation of vehicle routing related problem, we favor to select the direct coding for the LTCPP, since the

time-consuming encoding and decoding phase of the indirect coding can be avoided. As mentioned in the introduction of the long-term car pooling problem, a solution to the LTCPP includes dividing users into clusters (car pools) and identifying the route for each member in each cluster when this member acts as a server. Given that the problem concerns both clustering and routing, it is necessary to have all the information of each user in the representation, especially the routing is different when each user acts as a server. Thus, the representation is designed with two levels. The first level indicates the partition of users, while the second level records the routes of each member in the cluster when the member acts as a server. Moreover, in the second level, some other information is also collected and associated with each user, such as the total travel time and distance, the time schedule to pick up the cluster members when this user acts as a server, and whether a user is pooled with other users or travels alone.

Therefore, as abovementioned, in the first level of the proposed representation, the solution of dividing users into car pools is expressed. The representation consists in a set of clusters $S = \{P_1, P_2, ..., P_m\}$, and each cluster $P_i = \{C_j, C_k, ... , C_n\}$ is a sequence-non-important set includes the members assigned to a car pool. Note that the representation of each cluster may have different length, since the length is based on the number of members in the cluster.

In the second level, for each user C_j in each cluster, the following information is associated:

- The representation of the routing when this user acts as a server, this information is shown as a permutation R_j: $(C_j, C_k, ...)$ of users who are the pooled in C_j's car pool, started with C_j since his/her home is the start point of the route, then followed sequentially with the car pool members visited after.
- The representation of pickup times T_j: $(U_j, U_k, ...)$ for all his/her car pool members when this user acts as a server. The sequence follows the same order as the representation R_j. That is, the pickup time U_k in T_j corresponds to the user C_k in R_j. Note that the value U_j indicates the departure time of user C_j from his/her home.
- The Boolean value ϕ_i indicates if user C_j has been pooled with other user or travels alone.
- The value dis_j represents the total distance traveled by user C_j when this user acts as a server.
- The value tim_j corresponds to the total travel time of user C_j when this user acts as a server.
- The value arv_j shows the arrival time of user C_j at the destination when this user acts as a server.

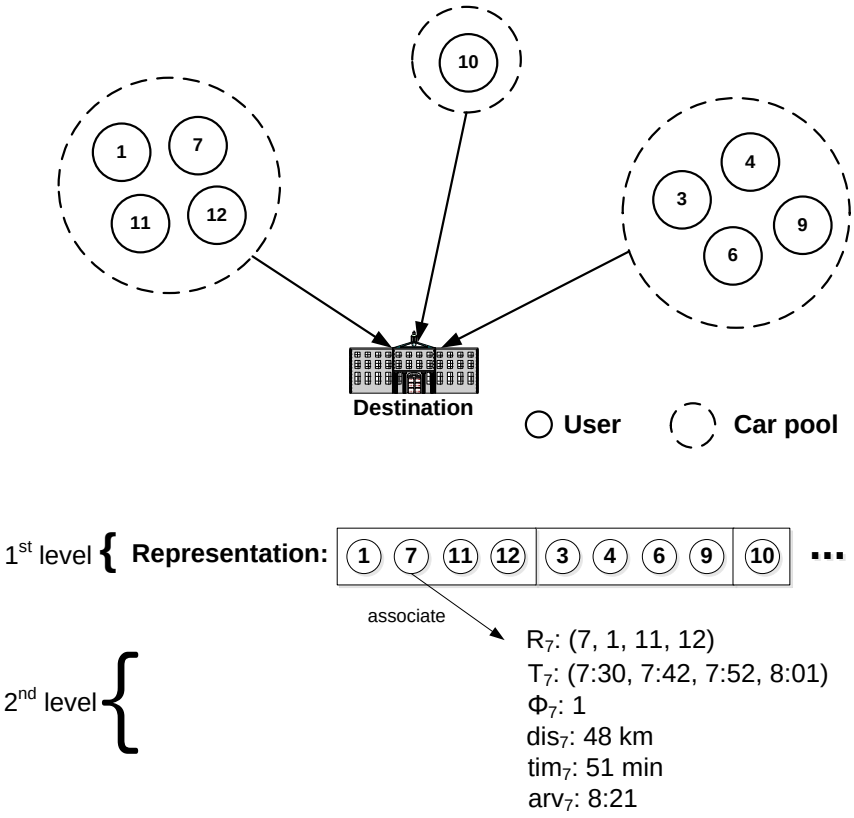


Figure 2.5: An example of the solution representation in VNS-LTCPP.

Therefore, this representation provides all the information of a user in the long-term car pooling problem. It offers and contributes in a clear manner to design long-term car pooling problem solutions, as shown in figure 2.5. A UML diagram of the solution representation can be found in appendix 3.

2.3.3 Initial Solution Heuristic

A reasonably structured initial solution will evolve to high quality solutions in a relatively small number of generations. In order to obtain a well-designed initial solution, we generate it with a two-steps approach introduced as follows.

The first step consists in selecting *n* users to construct *n* clusters. Each of these *n* users is considered as the representative of an individual cluster and the rest unselected users will be assigned to these clusters in the second step. The selection proceeds as follows. All users are put into a list with random order. A user *a* is selected randomly from the list to construct a cluster, and then user *a* and the *m* users nearest to user *a* are removed from the list, where *m* is an integer obtained by averaging the car capacity of all users. Afterwards, the second user *b* is also selected randomly from

the rest users in the list to construct a cluster, and then the user *b* and *b*'s nearest *m* users in the list are also removed from the list. The procedure ends when all the users are removed from the list. This mechanism avoids selecting representatives too close to each other to build clusters, so the clusters are guaranteed to be evenly distributed, which provides a good basis for the next step.

In the second step, a regret insertion procedure is applied. The procedure is based on the computation of a regret value for all users who are not selected as representative in the first step, in order to assign them to the constructed clusters. For each unselected user *i* and each representative *j* of a constructed cluster, the distance between *i* and *j* is computed by equation (2.1).

$$d_{ij} = \alpha \times \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + \beta \times \left| e_i + t_{ij} - e_j \right| \tag{2.1}$$

where x_i, y_i, x_j, y_j are the coordinates of the user *i* and user *j* respectively; e_i and e_j are the earliest departure time of user *i* and user *j*; α and β are adjusting parameters.

Then, the regret of user *i* is defined to be the difference between the shortest and second shortest distance computed above, as shown in equation (2.2).

$$regret_i = d_{ij} - d_{ik} \tag{2.2}$$

where d_{ij} and d_{ik} are the second shortest and the shortest geographical distances between *i* and the representatives of the clusters constructed in the previous step, respectively.

The construction algorithm tries to assign each unselected user to its nearest representative, considering the unselected users in the order of decreasing regrets. The users with the largest regret values are considered first, in order to avoid the large penalty associated with assigning him to the second nearest representative. Assignments are conditioned by car capacity, forbidding the assignment of users to the cluster which exceeds the car capacity of any member in this cluster. If the nearest representative is not available, the algorithm tries sequentially the next nearest representative until no available representative can be found. The procedure stops either when all the unselected users are assigned to the representatives or when it is impossible to assign any unselected user to any representative, then the unassigned the users are set to travel alone.

In order to reduce the complexity of the heuristic, the time window constraints are not checked during the assignment phase. Hence, a repair procedure is necessary to make sure the initial solution is feasible. After all car pools are built, the time windows when each user acts as a server are verified. The car pools violate the time windows will be divided into smaller car pools with minimum travel cost increase until no violations can be found.

Algorithm 2.3 presents the structure of the initial solution heuristic, the complexity of the algorithm is $O(n^2)$.

***Algorithm 2.3**: Initial Solution Heuristic.*

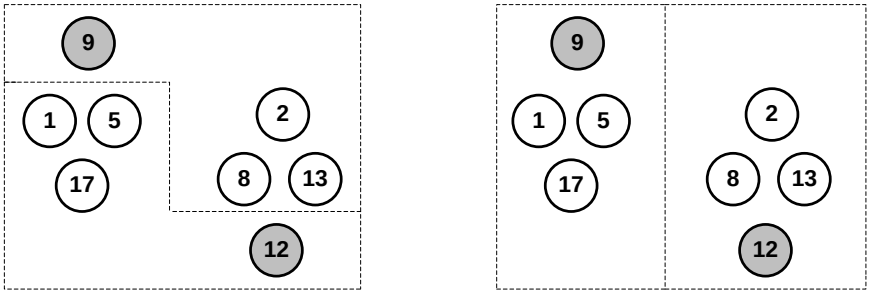
Compute the distance between each two users;
Put all users in a list;
***While** the list is not empty*
    Randomly select user k to build a new cluster and k is the representative;
    Remove user k from the list;
    Remove m k's nearest in-list users from the list;
End while
***For** each unselected user i;*
    Calculate the distances between user i and the representative of each cluster;
    Sort the representatives by increasing distance and record the result;
    $regret_i = d_{ij} - d_{ik}$; / j is the second nearest representative of a cluster, k is the nearest representative of a cluster*/*
End for
Sort the unselected users by decreasing regret value;
***For** each unselected user i*
    ***While** not all available representatives are tried*
        Assign the user to the nearest available representative;
    End while
    ***If** the user is not assigned*
        Set the user to travel alone;
End for
***For** each user i /* repair procedure */*
    Verify the time window when user i acts as a server;
    ***While** the time window is violated*
        Divide the current car pool(s) into two smaller car pools;
        Verify the time window of obtained car pools;
    End while
End for

2.3.4 Neighborhoods Design

In order to adapt VNS for our particular long-term car pooling problem, it is necessary to define the set of neighborhood structures and to establish the search procedure that will be applied to the solutions. All our neighborhoods and the search procedure are related to specific operators designed for the long-term car pooling problem. We propose four different neighborhoods $N_k(s)$ for our VNS-LTCPP algorithm. The neigh-

borhoods are used sequentially in the algorithm. Each neighborhood corresponds to an operator. The neighborhoods of our VNS- LTCPP approach are defined as follows.

Swap Neighborhood (N_1) is the set of solutions which results of the swap operator**.** It consists in swapping any user *i* with any user *j* who can pick up and deliver each of user *i*'s car pool members within his/her maximum driving time. The two selected users *i* and *j* are deleted from their original clusters and inserted into the each other's cluster. An example of the swap operator is shown in figure 2.6. User 9 and user 12 from different clusters are simultaneously placed into the other cluster. The complexity of this operator is $O(n^2)$.



(a) Before (b) After

Figure 2.6: An ideal example in Swap Neighborhood.

Chain Neighborhood (N_2) is the set of solutions obtained by the chain operator. The operator applies the ejection chain idea. A list of all the car pools is first initialized as following. A car pool *i* is randomly selected as the first element of the list, and then car pool *j* whose gravity center is the closest to the gravity center of car pool *i* is inserted to the list. Afterwards, another car pool *k* which is nearest to car pool *j* will be inserted. The procedure continues until all car pools are inserted into the list. Note that only the car pools which are not on the list can be inserted, so there will no repetitive car pools in the list. Then, the operator consists in selecting any car pool on the list as the start point and proceeding with the following procedures. Suppose the k^{th} pool of the list is selected, the user who is the farthest from the gravity center of the pool is moved to the $k+1^{th}$ pool of the list. Then, if the $k+1^{th}$ pool violates the car capacity constraint, the same procedure will be applied to it. The chain is designed to be cyclic, so the car capacity constraint is always satisfied. The gravity center of a car pool is calculated based on the distances calculated by equation (2.1) among the pool members regardless the destination. The chain operator allows moving the users without affecting well-clustered users. An example of chain neighborhood is shown in figure 2.7. Figure 2.8 presents the mechanism of the chain operator. The complexity of this operator is O(n).

Divide Neighborhood (N_3) is the set of solutions provided by the divide operator, the

operator consists in divide any car pool into two non-empty car pools with all possible combinations. An example of divide operator is presented in figure 2.9. The complexity of this operator is O(n).

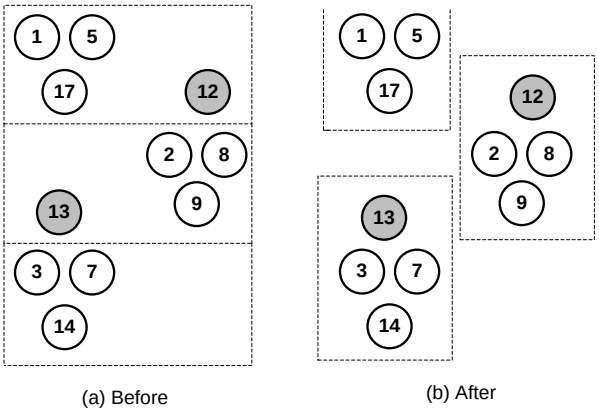


Figure 2.7: An ideal example in Chain Neighborhood.

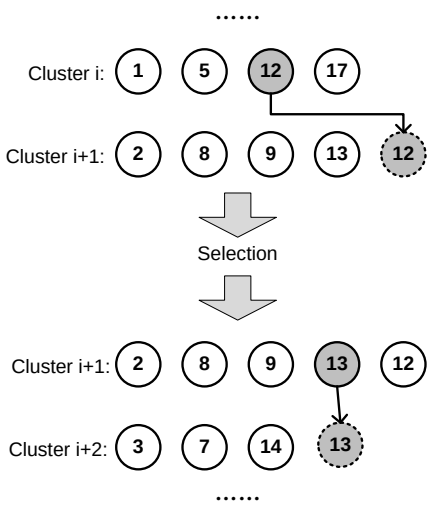


Figure 2.8: The mechanism of the Chain Operator.

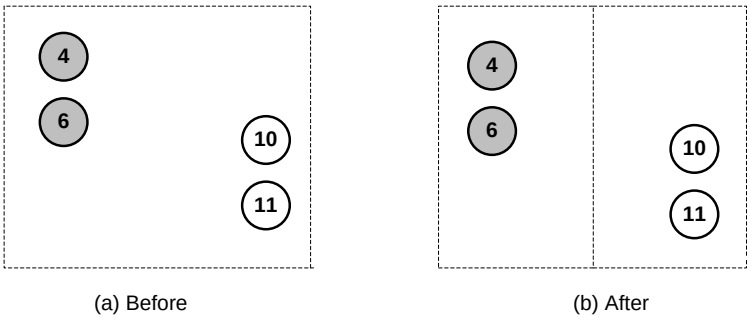


Figure 2.9: An ideal example in Divide Neighborhood.

Merge Neighborhood (N_4) is the set of solutions which results of the merge operator. The operator merges any two non-full car pools with respect to the car capacity constraint. Any non-full car pool *i* and non-full car pool *j*, which are able to satisfy the car capacity constraint after merging, are combined together by the operator. An example in merge neighborhood is presented in figure 2.10. The complexity of this operator is O(1).

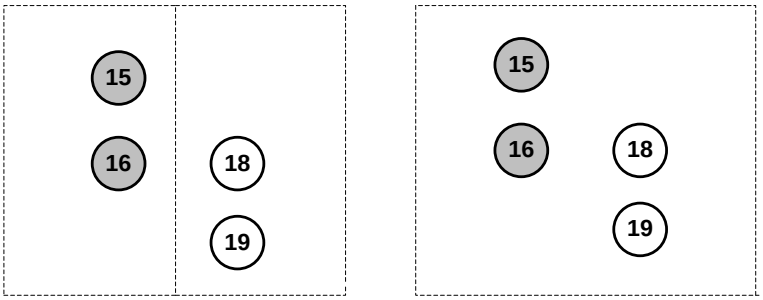


Figure 2.10: An ideal example in Merge Neighborhood.

2.3.5 Solution Evaluation

The evaluation of the objective function is usually the most expensive operation of any metaheuristic. For the long-term car pooling problem, a complete evaluation consists in calculating the cost and verify the car capacity and time window constraints for each car pool. Since the neighbor solutions of a solution *s* are only partially different from *s*, we provide a more efficient way to perform the evaluation of the solutions provided by the VNS-LTCPP and all our future approaches. Our evaluation proceeds based on the change *E(s, m)* of the current solution, where *s* is the current solution and *m* is the applied move. This is an important design in order to improve the efficiency and has been taken into account in the design of all our metaheuristics. It consists in evaluating only the transformation *E(s, m)* applied to a solution *s* rather than the complete evaluation of the neighbor solution *s'* where $s' = s + E(s, m)$. The definition of such an incremental evaluation and its complexity depends on the neighborhood used over the target optimization problem. In our case, the change only corresponds to at most two car pools for the Swap, Divide and Merge neighborhoods. For Chain neighborhood, the amount of related car pools varies. This evaluation mechanism can greatly increase the efficiency of our VNS-LTCPP approach, since the evaluation only focuses on a small amount of car pools.

Note that, the time window constraints are examined before the evaluation, a generated neighbor solution which is infeasible with respect to the time window will be repaired by the same procedure used in the construction of the initial solution.

2.3.6 Main structure of the VNS-LTCPP

As introduced above, our VNS-LTCPP consecutively makes use of four different neighborhoods of the current solution to achieve the variable neighborhood search structure. All orders of the neighborhoods are tested, and the one that yielded the best average output is chosen. The order of the neighborhoods applied is: Swap neighborhood, Chain neighborhood, Divide neighborhood and then Merge neighborhood.

The structure of VNS-LTCPP is given in the Algorithm 2.4. An initial solution is built according to the Regret Insertion heuristic. Then, the algorithm selects a random solution from the current neighborhood and applies the corresponding operator to obtain a local optimum. For each operator, the first improvement policy is applied. That is, a move is performed as soon as the solution cost decreases. Note that the moves violating the time window constraint are repaired before the evaluation, so the obtained local optimal is convincing. Afterwards, if the resulting solution is better than the incumbent, the algorithm moves to the resulting solution, and restart the search with the first neighborhood of the incumbent; otherwise the algorithm switches to the next neighborhood.

The optimization procedure stops when no neighborhood is capable of improving the current solution or the maximum iteration number has been reached.

***Algorithm 2.4**: The VNS-LTCPP Approach.*

```
Define the set of neighborhood Nk (k = 1, ... , n );
s0 = RegretInsertionInitialSolution();      /* Generate initial solution */
s = s0;
While the stop criteria is not met
    k = 1;    /* Start with the first neighborhood */
    While k does not exceed n;
        s' = pickRandom(Nk(s));    /* Select a solution from current neighborhood */
        s'' = MoveOperator(s');    /* Apply operator to obtain local optimum*/
        If Q(s'') < Q(s)
            s = s'';
            k = 1;
        Else
            k = k + 1;
    End while
End while
```

2.4 Experimental Results and Analysis

In order to provide an exclusive evaluation of the performance of our algorithm, our experimental results are compared with the simulation-based approach (SB) [Correia and Viegas, 2008]. Selecting the SB approach is because this approach solves the problem with the aid of the optimization software, which gives it high probability in providing high solution quality. As mentioned in the related works in chapter 1, the simulation-based method is a divide-and-conquers approach. The process starts with using the K-means clustering algorithm to divide all the users into small clusters such that each small cluster can be processed by the optimization program in an acceptable period of time. Then, all the small clusters are sent to the optimization program to search for the possible group combinations. The people that were not able to find a match in the previous iteration are set together for another iteration.

Since the benchmarks and implementation environments of this work are different from ours, in order to provide a fair and convincing comparison, we implemented the approach in our experiment environment and use it to solve our benchmarks presented previously. Note that the approach is implemented exactly as it is described in the literature with no modification.

The two algorithms were implemented in our algorithm test and analysis platform with JAVA, and all results were obtained running the code on a Windows Operating System with Intel Core i7 740QM 2.9 GHz CPU and 4 GB of RAM. Each algorithm has been executed 30 times for each instance. The state-of-the-art metaheuristics normally use the CPU time or total amount of iteration as stopping criterion. Since in the LTCPP, the CPU time and total amount of iterations vary for instances with different sizes, the SB approach uses a stop criteria when no new car pool can be formed. In corresponding with the SB approach, our approach ends when there is no improvements can be found.

Table 2.1 compares the experimental results of the C set instances. The VNS-LTCPP outperforms the SB approach on 6 instances considering the average solution quality of 30 runs. Furthermore, the VNS-LTCPP's best found solutions of 7 instances are also better than the ones provided by the SB approach. Observe that, the execution of VNS-LTCPP is much less time-consuming than the SB approach, which denotes that our algorithm is reactive and is able to reach competitive solutions in a short time.

Table 2.2 shows the percentage the VNS-LTCPP outperforms the SB approach on set C instances, in the aspects of average solution quality and computing time. For each instance, the outperforming percentage is calculated as *(SB's value – VNS-LTCPP's value) / SB's value*. Each value in table 2.2 is obtained by averaging the outperforming percentages of the three same-size instances. Comparing with the SB approach, the VNS-LTCPP can provide better solution quality in 13.6% of the computing time of the SB approach (1– 86.4% from table 2.2).

Inst	Size	VNS-LTCPP			SB [Correia, 2007]		
		Best	Avg	Time(s)	Best	Avg	Time(s)
C101	100	1644.9	1684.6	7	1647.4	1669.2	91
C102	100	1729.2	1753.8	9	1717.5	1724.8	94
C103	100	1545.9	1563.6	7	1532.2	1599.4	85
C201	200	2676.9	2723.8	24	2761.7	2868.6	329
C202	200	3070.7	3145.2	26	3081.7	3114.1	473
C203	200	2935.1	2993.7	37	2975.1	3182.4	394
C401	400	6024.5	6130.1	255	6174.2	6860.3	934
C402	400	5055.9	5110.7	197	5383.7	5524.5	683
C403	400	6079.6	6322.5	287	6675.2	6994.5	1257
Total		30762.7	31428.0	849	31948.7	33537.8	4340

Table 2.1: Experimental results of set C instances (clustered user distribution).

Set	Size	Best	Average	Time
C	100	-0.47%	-0.12%	92.92%
	200	1.59%	3.33%	92.61%
	400	5.81%	9.25%	73.67%
Avg		2.31%	4.15%	86.40%

Table 2.2: Solution quality and computing time comparison on set C instances.

Ints	Size	VNS-LTCPP			SB [Correia, 2007]		
		Best	Avg	Time(s)	Best	Avg	Time(s)
R101	100a	2211.2	2286.6	8	2235.1	2265.4	100
R102	100b	1856.7	1898.7	10	1832.8	2091.7	97
R103	100c	2288.3	2379.8	7	2204.7	2418.5	80
R201	200a	4349.2	4464.6	35	4425.0	4567.1	430
R202	200b	3970.3	4162.0	27	3952.4	4283.3	231
R203	200c	4118.5	4282.5	36	4092.4	4257.5	540
R401	400a	8097.1	8398.4	316	8787.8	8993.8	1106
R402	400b	6411.8	6975.0	196	7258.7	7417.5	896
R403	400c	8312.1	8422.0	309	8841.9	8933.5	1037
Total		41615.2	43269.6	944	43630.8	45228.3	4517

Table 2.3: Experimental results of set R instances (random user distribution).

Table 2.3 presents the experimental results on the R set instances. The random distributed instances give the VNS-LTCPP some difficulties in constructing a good initial solution. However, the same problems are met by the K-means algorithm of the SB approach. On this set of instances, the VNS-LTCPP outperforms the other approach on 7 instances in average solution quality and 5 instances in best found

solution. Table 2.4 reveals the percentage of outperforming in the same manner as table 2.2. The computing time of the VNS-LTCPP is still significantly less than the other approach.

Set	Size	Best	Average	Time
R	100	-1.34%	3.30%	90.98%
	200	0.21%	1.50%	91.17%
	400	8.51%	6.10%	73.25%
Avg		2.46%	3.63%	85.13%

Table 2.4: Solution quality and computing time comparison on set R instances.

Inst	Size	VNS-LTCPP			SB [Correia, 2007]		
		Best	Avg	Time(s)	Best	Avg	Time(s)
W101	100	866.7	886.9	7	864.2	885.7	81
W102	100	1007.0	1041.5	8	1007.5	1037.9	82
W103	100	1118.9	1173.8	8	1100.5	1187.6	87
W201	200	1614.4	1682.5	38	1717.1	1722.9	341
W202	200	1943.5	2003.6	25	2014.3	2127.7	406
W203	200	1733.7	1806.8	44	1860.5	1965.1	371
W401	400	2975.2	3076.8	520	3168.4	3442.8	955
W402	400	3625.6	3713.0	306	3633.5	3984.9	792
W501	565	5105.2	5288.4	578	5377.5	5858.3	1621
Total		19990.1	20673.3	1534	20743.5	22212.9	4736

Table 2.5: Experimental results of set W instances (real world cases).

The experimental results for the set W instances (real-world instances) are presented in table 2.5. The VNS-LTCPP provides better solution quality on 7 instances in both average and best found solution. The outperforming percentage is shown in table 2.6.

Set	Size	Best	Average	Time
W	100	-0.63%	0.23%	90.80%
	200	5.44%	5.41%	90.28%
	400	5.63%	9.06%	57.09%
Avg		3.48%	4.90%	79.39%

Table 2.6: Solution quality and computing time comparison on set W instances.

Considering all three sets of instances, the solution quality of VNS-LTCPP is significant better than the other approach when dealing with the instances with 200 and 400 users in all three sets. Moreover, the computing process of VNS-LTCPP is

much less time-consuming than the SB approach, which makes the VNS-LTCPP a good candidate for the real-world application.

Instance	VNS-LTCPP			SB [Correia, 2007]		
	Avg	R	R^2	Avg	R	R^2
C101	1684.6	2	4	1669.2	1	1
C102	1753.8	2	4	1724.8	1	1
C103	1563.6	1	1	1599.4	2	4
C201	2723.8	1	1	2868.6	2	4
C202	3145.2	2	4	3114.1	1	1
C203	2993.7	1	1	3182.4	2	4
C401	6130.1	1	1	6860.3	2	4
C402	5110.7	1	1	5524.5	2	4
C403	6322.5	1	1	6994.5	2	4
R101	2286.6	2	4	2265.4	1	1
R102	1898.7	1	1	2091.7	2	4
R103	2379.8	1	1	2418.5	2	4
R201	4464.6	1	1	4567.1	2	4
R202	4162	1	1	4283.3	2	4
R203	4282.5	2	4	4257.5	1	1
R401	8398.4	1	1	8993.8	2	4
R402	6975	1	1	7417.5	2	4
R403	8422	1	1	8933.5	2	4
W101	886.9	2	4	885.7	1	1
W102	1041.5	2	4	1037.9	1	1
W103	1173.8	1	1	1187.6	2	4
W201	1682.5	1	1	1722.9	2	4
W202	2003.6	1	1	2127.7	2	4
W203	1806.8	1	1	1965.1	2	4
W401	3076.8	1	1	3442.8	2	4
W402	3513	1	1	3984.9	2	4
W501	5288.4	1	1	5858.3	2	4
Avg		1.25			1.74	
Sum		34	48		47	87

Table 2.7: Friedman test between the VNS-LTCPP and the SB approach.

A further evaluation of the performance among the two approaches has been conducted by using a Friedman test [Friedman, 1940]. The test consists in the average solution quality of the two approaches on all 27 instances used in our experiments. The detail information is presented in table 2.7, the detail calculation can be found in appendix 2. The Friedman statistic value *T* of table 2.7 is calculated to be *7.85*, while the threshold for the F distribution with a significance level 0.01 is *7.72*. Since *T* is greater than the threshold, it is proven that the VNS-LTCPP's performance is significant better than the SB approach.

The accuracy of the VNS-LTCPP is examined by calculating the standard error (column Std) of the solutions obtained in 30 runs of each instance. The solution quality difference (column Diff) between the best found solution quality and the average solution quality of *each* instance in the previous tables is also calculated. Table 2.8 shows the *average* of the abovementioned values of the three same-size instances. The average differences between the best found solution and the average solution of the three sets of instances are 2%, 3.6% and 3.4%, respectively, which indicates the VNS-LTCPP approach can be considered to be accurate for a trajectory-based metaheuristic.

Size	C set instances				R set instances				W set instances			
	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)
100	1640.0	1667.3	12.1	1.6	2118.7	2188.4	38.9	3.2	997.5	1034.1	18.4	3.5
200	2894.2	2954.2	22.7	2.0	4146.0	4303.0	86.2	3.6	1763.9	1831.0	32.1	3.7
400	5720.0	5854.4	79.2	2.3	7607.0	7931.8	121.4	4.1	3835.3	3959.4	67.3	3.1
Avg	3418.1	3492.0	38.1	2.0	4623.9	4807.7	82.2	3.6	2198.9	2274.8	39.3	3.4

Table 2.8: Evaluation of the accuracy of the VNS-LTCPP.

2.5 Conclusion

This chapter presented a Variable Neighborhood Search approach for solving the long-term car pooling problem. The main characteristic of this approach consists in the ability of moving from one neighborhood to another one throughout the optimization process. This ability offers an adaptive mechanism for tracking the optimum in the search space. For this proposal, different neighborhoods have been well designed to achieve a good search ability of the approach.

The experiments of VNS-LTCPP have been performed on three sets of structurally different instances. Each set includes large scale instances. The experimental results are compared with the SB approach, which is a hybrid approach of K-means and CPLEX.

The VNS-LTCPP is able to yield better results for a large number of instances. It provides remarkable performance especially on large scale instances. Comparing the computing time of the two approaches, the CPU time cost to obtain a solution by VNS-LTCPP are much less than the SB approach. Hence, the VNS-LTCPP approach can work as a useful method to solve the long-term car pooling problem.

According to the literature of optimization, population-based metaheuristics are usually considered to be more effective in diversification ability than trajectory-based ones. However, in terms of intensification search, the trajectory-based metaheuristic is known as the more effective method. In general, the degree of success of these meth-

ods on a given problem depends largely on their ability to strike a balance between exploration and exploitation. The ability of population-based metaheuristics to sample the search space and the fact that they simultaneously manipulate a group of solutions increase their potential solving ability for complex optimization problems.

Therefore, in our next step, population-based metaheuristics are applied to solve the long-term car pooling problem. Considering the effectiveness the VNS-LTCPP approach, the operators will be adopted to the local search procedures of some of our population-based approaches which will be reviewed in the future chapters.

CHAPTER 3

Swarm Intelligence Metaheuristics for the Long-term Car Pooling Problem

CONTENTS

Abstract

In this chapter we present a Clustering Ant Colony Algorithm to solve the LTCPP. The algorithm belongs to the swarm intelligence family, and is proven to have good ability to provide high quality solutions for the LTCPP instances. Experiments are conducted to verify the performance of the algorithm.

3.1 Introduction

During the exploration to the search space, there usually remains some information from the past that can be used in the future. As mentioned in the previous chapter, the trajectory-based metaheuristics manage only one solution during iterations, and the solution is always obtained based on the one from the previous iteration. This characteristic not only limits the diversification search ability of the trajectory-based metaheuristics, but also forbids the useful composition of the former solutions being passed to future solutions. Therefore, it is necessary to apply a methodology to track good solutions through the exploration in the search space. The required algorithm should not only be able to improve the solution in hand, but should also be able to conserve the good composition of the former solutions and use them in the future search.

Population-based metaheuristics exhibit a number of potential advantages for such purposes. Both swarm intelligence family and evolutionary family of the population-based metaheuristics have been considered by us to be applied in solving the LTCPP. In this chapter, the swarm intelligence family will be discussed, and the evolutionary family will be presented in the next chapter.

In the swarm intelligence family, we choose the Ant Colony Optimization (ACO) as the fundamental structure of our first population-based algorithm, since the ACO has a more flexible paradigm comparing with other swarm intelligence algorithms. Grounding on the characteristic of the LTCPP, which is a combination of clustering and routing, we believe ACO is a suitable for solving this problem based on its good exploration ability and flexible pheromone representation.

In recent years, there have been increasing interests in using ACO in dealing with vehicle routing problems. The underlying principle of ACO is based on swarm intelligence, hence it is expected to be capable of self-organization and be able to adapt to difference problems. In addition, ACO has been proven to be able to find good solution fast in the search space due to its ability to store and exploit previous solutions. One of the most appealing features is that, the behavior which the ants move among the nodes in a graph can be adapted into both the clustering operation and the routing operation of the LTCPP. Thus, it is possible to merge the clustering operation and the routing operation together by using the ACO structure. For this purpose, we present in this chapter a Clustering Ant Colony Algorithm (CAC) for solving the Long-term Car Pooling Problem. The main idea of the CAC is to cluster the car pool members of each car pool during the construction of the ant's path. Furthermore, the clustering experiences are memorized to direct the search of the future ants. Thus, the classic ACO algorithm has been transformed into a method with both clustering and routing abilities. This chapter starts with an introduction to the swarm intelligence metaheuristics and the common concepts related to these approaches. In Section 3.3 the classic ant colony algorithm is introduced. Section 3.4 presents our Clustering Ant Colony Algorithm for solving the LTCPP. The section consists in the preference and attrac-

tiveness representations, the preference update mechanism, and the embedded local search procedure. In Section 3.5 our CAC approach is tested on the sets of benchmarks we developed. In addition, the performances of our algorithm are assessed and discussed. Finally, in section 3.6, we conclude this chapter with a summary and introduce the future step of this thesis.

3.2 Swarm Intelligence Metaheuristics

Swarm intelligence (SI) refers to the collective behavior of self-organized, decentralized systems. The expression was introduced by Beni and Wang [1989], in the context of cellular robotic systems. The inspiration of the swarm intelligence often comes from nature, especially biological systems. Many different kinds of swarms in the nature can perform some collective behavior without any individual controlling the group, or being aware of the overall group behavior. Although lacking individuals in charge of the group, the swarm as a whole can show high intelligent behaviors. This is the result of the interaction of spatially neighboring individuals that act on the basis of simple rules.

SI metaheuristics typically consist in a population of simple agents interacting locally with their environment. The agents act on simple rules, and there is no centralized control structure dictating how individual agents should behave. However, the collective behaviors of agents interacting locally with their environment cause coherent functional global patterns to emerge. An intelligent global behavior unknown to the individual agents thus is generated. SI metaheuristics provide a basis with which it is possible to explore collective or distributed problem solving without centralized control or the provision of a global model. Thus, the characterizing property of swarm intelligence metaheuristic is its ability to act in a coordinated way without the presence of a coordinator or of an external controller.

The other common characteristics of the swarm intelligence metaheuristics can be categorized as follows.

- They are composed of many individuals. The individuals are relatively homogeneous.
- The behavior of each individual is described in probabilistic terms. Each individual has a stochastic behavior that depends on its local perception of the neighborhood.
- The interactions among the individuals are based on simple behavioral rules that exploit only local information that the individuals exchange directly or via the stigmergy.
- The overall behavior of the system is obtained from the interactions of individu-

als with each other and with their environment, which provides a self-organized system.

Because of the above properties, the swarm intelligence metaheuristics are scalable, parallel, and fault tolerant.

Scalability means that a system can maintain its function while increasing its size without the need to redefine the way its individuals interact. Because in a swarm intelligence system, the individuals' behavior is only influenced by the neighboring information exchange or the stigmergy, the number of interactions tends not to grow significantly when the overall number of individuals in the swarm increases. In metaheuristics, scalability is important because a scalable system can increase its performance by simply increasing its size, without the need for any reprogramming.

Parallel action is possible in swarm intelligence metaheuristic because individuals composing the swarm can perform different actions in different places at the same time. In metaheuristics, parallel action is desirable because it can help to make the system more flexible and provide faster computing speed by taking care simultaneously of different aspects of a complex task.

Fault tolerance is an inherent property of swarm intelligence metaheuristics due to the decentralized, self-organized nature of their control structures. Because the system is composed of many interchangeable individuals and none of them is in charge of controlling the overall system behavior, a failing individual can be easily dismissed and substituted by another one that is fully functioning. Also, the effect of the individuals which provide low quality solutions can be minimized.

The well-known swarm intelligence metaheuristics include the ant colonies optimization (ACO), the artificial bee colony algorithm (ABC), and the particle swarm optimization (PSO).

The ant colony optimization (ACO) [Dorigo and Stützle, 2004] is a class of optimization algorithms inspired by the actions of an ant colony. ACO methods are useful in problems that require constructing paths. To use ACO, the given optimization problem needs to be transformed into the problem of finding the minimum cost path on a weighted graph. Then, a set of agents called "artificial ants" can search for good solutions in the transformed graph. The artificial ants incrementally build solutions by moving on the graph. The solution construction process is stochastic and is biased by a pheromone value, that is, a set of parameters associated with the nodes or edges of the graph. The values are updated when the ants obtained new solutions. ACO has been successfully applied to many combinatorial optimization problems, as well as to discrete optimization problems that have stochastic or dynamic components. Examples are the application to vehicle routing problems and to the probabilistic traveling salesman problem. Ant colony optimization is probably the most successful example of the swarm intelligence metaheuristics with numerous applications to real-world problems. One variation on this approach is the artificial bee colony algorithm, which is more analogous to the foraging patterns of the honey bee.

Artificial Bee Colony (ABC) algorithm is a swarm intelligence based metaheuris-

tic introduced by Karaboga [2005], and simulates the foraging behavior of honey bees. The agents of ABC algorithm are assigned with three different roles: scout bee, leader bee and follower bee. In the beginning, the algorithm starts with the scout bees being placed randomly in the search space. Then, the fitnesses of the solutions provided by the scout bees are evaluated. The scout bees that have the highest fitnesses are chosen as leader bees, and some of the other scout bees are assigned to each leader bee to be their follower bees. The follower bees are intended to generate neighborhood searches to the solutions of provided by the leader bees. Searches in the neighborhood of the best solutions which represent more promising solutions are made more detailed by recruiting more follower bees than the other leader bees. Together with scouting, this differential recruitment is a key operation of the Bees Algorithm. At last, the remaining scout bees in the population are assigned randomly around the search space scouting for new potential solutions. These steps are repeated until a stopping criterion is met. The ABC algorithm has a well-balanced exploration and exploitation ability, since the different types of bees are able to maintain the intensity and the diversity at the same time.

Particle swarm optimization (PSO) [Kennedy and Eberhart, 1995] is another swarm intelligence based stochastic optimization technique for the solution of continuous optimization problems. The algorithm is inspired by social behaviors in flocks of birds and schools of fish. The problems to be solved by PSO have to be transformed into an n-dimensional space where the best solution has to be represented as a point or surface in the space. A set of agents called particles is used to search for good solutions to the given optimization problem. Each particle is a solution of the considered problem and uses its own experience and the experience of neighbor particles to choose how to move in the search space. In practice, in the initialization phase each particle is given a random initial position and an initial velocity. The position of the particle represents a solution of the problem and has therefore a value, given by the objective function. While moving in the search space, particles memorize the position of the best solution they found. At each iteration of the algorithm, each particle moves with a velocity that is a weighted sum of three components: the old velocity, a velocity component that drives the particle towards the location in the search space where it previously found the best solution so far, and a velocity component that drives the particle towards the location in the search space where the neighbor particles found the best solution so far. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large numbers of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

3.3 Classic Ant Colony Algorithm

As abovementioned, the ant colony optimization algorithm (ACO) is a well-known metaheuristic for solving computational problems which can be reduced to finding good paths through graphs. This algorithm was initially proposed by Dorigo [1992] in his doctoral thesis with the aim to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food.

In the natural world, the real ants move randomly at the beginning of their search for food. When the food source is found and the ants return to their colony, they lay down pheromone trails on the path they traveled. If other ants find such a trail, they are likely to follow the trail instead of continuing to travel randomly. Moreover, while returning to the colony, the ants reinforce the trail if they eventually find food corresponding to this trail.

The pheromone trail evaporates over time, results in reducing its attractive strength. The more time it takes for an ant to travel down the path and return to the colony, the more time the pheromones have to evaporate. Hence, a relatively short path will be travelled over more frequently by the ants, and thus the pheromone density becomes higher on shorter paths than longer ones. So after a period of time, most of the ants will be attracted to the shorter paths. Another advantage of the pheromone evaporation is to avoid the convergence to a locally optimum. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones, and the attraction would only increase through time. In that case, the exploration of the solution space would be constrained.

Thus, when one ant finds a short path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve.

The environment is used by the ants as a medium of communication. They exchange information by depositing pheromones instead of communicating directly with each other. The information exchanged has a local scope, only an ant located where the pheromones were left has a notion of them. This system is called "Stigmergy" and occurs in many social animal societies. There are two kinds of feedbacks in the system: positive feedback where the deposit of pheromone attracts other ants that will strengthen it and negative feedback where dissipation of the route by evaporation prevents the system from thrashing. Theoretically, if the quantity of pheromone remained the same over time on all edges, no route would be chosen. However, because of feedback, a slight variation on an edge will be amplified and thus allow the choice of an edge. The algorithm will move from an unstable state in which no edge is stronger than another, to a stable state where the route is composed of the strongest edges.

In the ACO algorithm, the same behavior as the nature ants has been followed. The basic philosophy of the algorithm involves the movement of a colony of ants

through the different states of the problem influenced by two local decision policies, pheromone trail and attractiveness. The pheromone trail refers to the similar concept in nature ant colony, while the attractiveness is the physical attraction of a state. Typically, the further the state is from the ant, the less attractiveness it can provide. Thereby, each artificial ant incrementally constructs a solution for the problem by iteratively adding solution components to the partial solution. When an ant completes a solution, the ant evaluates the solution and modifies the pheromone trail value on the components used in its solution. This pheromone information will direct the search of the future ants. Furthermore, the pheromone trail evaporates through time, and the evaporation reduces all trail values thereby avoiding any possibilities of the ants being trapped in a local optimum. The general structure of the ACO is shown in Algorithm 3.1.

Algorithm 3.1*: Ant Colony Optimization Metaheuristic.*

Initial the attractiveness;

While *not Termination Criterion (*s_t *)* ***do***
    For *k = 1, m (m = number of ants)* ***do***
        While *a complete solution is not obtained* ***do***
            compute the probability to move to other states;
            choose the state j to move into;
            memorize the state j to the list of visited states;
        End while
        evaluate the solution;
        update pheromone;
    End for
End while

As presented in Algorithm 3.1, each ant starts from its initial state and moves from its current state to a new state *j* corresponding to a more complete partial solution. The partial solution refers to one of the solution states.

A roulette wheel selection is normally applied to the ants for probabilistically choosing the state to move. As abovementioned, the probability of the selection is influenced by two decision policies, which are the attractiveness of move, indicating the a priori desirability of that move and the pheromone trail, indicating the experiences left by the previous ants. The higher the pheromone and attractiveness associated to an expansion, the higher the probability an ant will move to it.

The k_{th} ant moves from state *i* to state *j* with the probability calculated in equation (3.1).

$$p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{n \in N}(\tau_{in})^{\alpha}(\eta_{in})^{\beta}} \tag{3.1}$$

where N is the set of all possible moves between two states, τ_{ij} is the value of pheromone deposited on the move from state *i* to *j*, $\alpha \geq 0$ is a parameter to control the influence of τ_{ij}, η_{ij} is a priori knowledge of the move from state *i* to *j* and $\beta \geq 0$ is a parameter to control the influence of η_{ij}. The η_{ij} is typically calculated as the reciprocal of d_{ij}, where d_{ij} is the distance between state *i* and state *j*, which refers to the cost of transition between the two states.

Once an ant completes its solution, it evaluates the solution, and then deposits the pheromone trails on the moves it took, so each move used by the ant receives an increasing of additional pheromone proportional, the value corresponds to the quality of the solution of the ant. In some ACO approaches, this procedure is done until all ants finish their solutions.

Another important behavior in the classic ACO is the pheromone evaporation, which means all the pheromone values are iteratively decreased by a certain percentage. This behavior decreases the intensity of the pheromone value which avoids a fast convergence towards the local optima. The system is designed to favor the new deposited pheromone information instead of the old ones.

The pheromone values are updated by equation (3.2).

$$\tau_{ij} = (1-\rho) \times \tau_{ij} + \Delta\tau_{ij}^{k} \tag{3.2}$$

where ρ is the pheromone evaporation coefficient and $\Delta\tau_{ij}^{k}$ is the additional amount of pheromone deposited. In a TSP, $\Delta\tau_{ij}^{k}$ is normally calculated as equation (3.3).

$$\Delta\tau_{ij}^{k} = \begin{cases} Q / L_k & \text{if } arc(i,j) \text{ is used by ant } k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where L_k is the cost of the k_{th} ant's tour and Q is a constant.

It is difficult to define precisely what algorithm is or is not an ant colony approach, because the definition varies according to the authors and applications. Generally speaking, each solution of the ACO is generated by an ant moving in the search space. Ants leave their pheromone trails on their solutions and in the meantime attracted by the previous pheromone information left by the ancestors. They can be seen as probabilistic multi-agent algorithms using a probability distribution to make the transition between iterations. The characteristic which distinguishes ACO algorithms

from other relatives, such as algorithms to estimate the distribution or particle swarm optimization, is precisely their constructive aspect.

We introduce some of most popular variations of ACO Algorithms as follows.

- Elitist ant system

  The global best solution deposits pheromone on every iteration along with all the other ants.

- Max-Min ant system

  The Maximum and Minimum pheromone amounts are added to the system. Only the global best or the iteration best solution deposits pheromone.

- Rank-based ant system

  All solutions are ranked according to their quality. The amount of pheromone deposited is then weighted for each solution, such that solutions with better quality deposit more pheromone than the ones with lower quality.

- Continuous orthogonal ant colony

  By implementing an "adaptive regional radius" method, the algorithm can reduce the probability of being trapped in local optima and therefore enhance the global search capability and accuracy.

3.4 Clustering Ant Colony Algorithm for LTCPP

During our following research with the ant colony algorithm, we first tried using a “route first, cluster second” approach where the ACO is applied to search the shortest path connecting all users, then the path is divided into car pools according to car capacity and time window constraints; we also implemented a “cluster first, route second” approach where we use K-means algorithm to cluster the users into car pools, then construct route for each car pool. But the research reveals that the separation of routing and clustering leads to lots of difficulty in finding the good solution. No matter which mechanism is applied first, it limits the search space, to such an extent that it is very hard for the second applied mechanism to find a high quality solution. So in order to provide an effective and efficient method for LTCPP, in our Clustering Ant Colony Algorithm, the ant is vested the ability of clustering during its tour and memorizes its clustering experience to direct the search of the future ants. Thus, the classic ACO algorithm has been transformed into a method with the ability of both clustering and routing. In this section, we introduce the structure and the concepts of the Clustering Ant Colony Algorithm (CAC) for the LTCPP.

3.4.1 Main Structure of the CAC

In the CAC, the ants are able to cluster the users during its tour and memorize its clustering experience to direct the search of the future ants. To direct the clustering activity of the ant, we introduce a preference concept into the ant system to replace the traditional pheromone trail. The preference information is defined as the preference of pooling two users in the same cluster. When an ant starts a tour from a user, it starts to build a car pool in the meantime. The ant then behaves according to a roulette wheel selection based on the preference and the attractiveness. It can either visit a new user and insert him into its current car pool or end the current car pool and select a new user to start a new car pool. When all the users are visited by the ant, the tour of the ant is considered finished. After all ants finish their tour in the current iteration, several solution with the best fitness value is selected to be applied a local search procedure. At the end of iteration, the preference information between the users in the same car pool of each selected solution will be increased. By this mechanism, the clustering experience is always memorized and updated to direct the search of future ants. The general structure of the CAC is specified as following Algorithm 3.2.

Algorithm 3.2*: Clustering Ant Colony Algorithm.*

Initialize preference and attractiveness.

While *the stop criteria is not met* ***do***

  For *k =1, k ≤ the number of ants* ***do***

    Repeat

      (a) *Select a new user and build a new car pool;*

      (b) *Insert the current user into the current car pool;*

      (c) *Check car capacity constraint:*

        If *the car capacity is reached, go to step (a);*

        Else *continue;*

      (d) *Select next activity in probability based on preference and attractiveness:*

        If *visiting another user is chosen, check time window constraint:*

          If *time window is satisfied then go to step (b);*

          Else *go to step (a);*

        Else if *ending current car pool is chosen, go to step (a);*

    Until *all users are visited;*

  End for

  Select m best solutions;

  Apply local search;

  Update the preference values based on the composition of selected solutions;

End while

The first step of the algorithm is to initialize the preference and attractiveness information. In the second step, the ant starts to construct a solution. The ant will first select a new user to start a new car pool, then if the car capacity and time window constraints are satisfied, it will continue search for pool members according to the probability based on the preference and attractiveness information. During the construction of the car pool, if the ant decides to end the current pool or the car capacity or time window constraints are violated, the ant will stop the current pool, and stochastically search for a new user to start a new pool. The probability of finding a user is calculated based on the attractiveness information. Until all ants finish building their solution, several best solutions among them will be selected to be applied a local search procedure. The detailed structure and sequence of the local search will be introduced in section 3.4.5. At the end of iteration, the composition of the solutions will be used to update the preference information. Until the stop criteria are met, the algorithm will go to the next iteration.

3.4.2 LTCPP Solution Representation

A representation for the solution has to be designed in order to provide a suitable mapping between our problem and the CAC algorithm. As discussed in chapter 2, we favor to use the direct coding for the LTCPP, since the time-consuming encoding and decoding phase of the indirect coding can be avoided. In the CAC, the ants proceed with both clustering operation and routing operation. Therefore, it is necessary to cover both the clustering information and the routing information of each user in the representation. Thus, the representation is still designed with two levels as in our Variable Neighborhood Search. The first level indicates the partition of users, while the second level records the total travel time and distance, the route and time schedule to pick up the cluster members when each user acts as a server. A detailed introduction can be found in section 2.3.2 of chapter 2.

3.4.3 Preference Information

Definition 3.4.1
Between any user i and user j in the long-term car pooling problem, a non-negative value w_{ij} is associated. The value reveals the existing experience on the frequency of user i and user j being pooled in the same car pool. The value is named preference information, since it refers to the preference of pooling user i and user j together.

The preference information can be considered as variant pheromone information. It is stored in an $n \times n$ matrix where $n = |U|$ is the number of users in an instance. The weight values of the matrix indicate the preference level between each two users to be pooled together, as shown in figure 3.1.

Note that, in the preference matrix of the CAC, the value corresponding to a user *i* and the user itself is also calculated. This value is defined according to definition 3.4.2.

Definition 3.4.2

For any user i in the long-term car pooling problem, a non-negative value w_{ii} is associated. The value indicates the level of user i's tolerance level of being pooled with other users. The value can be considered as preference of a user to himself/herself, and it shows the will of user to avoid new members being inserted to his/her car pool.

Preference Matrix

	U1	U2	U3	U4	U5	U6
U1	*1.3*	*0.5*	*0.1*	*0*	*1.5*	*0*
U2	*0.5*	*1.8*	*2.4*	*2.7*	*0*	*0*
U3	*0.1*	*2.4*	*1.5*	*0.2*	*0*	*0.9*
U4	*0*	*2.7*	*0.2*	*1.7*	*0.8*	*1.2*
U5	*1.5*	*0*	*0*	*0.8*	*2.1*	*1.7*
U6	*0*	*0*	*0.9*	*1.2*	*1.7*	*1.4*

Figure 3.1: An example of the preference matrix.

In the initialization of the preference information, the time window constraints (3.4) to (3.7) are checked. The constants in the equations are the same as the ones in the mathematical model in chapter 1. Constraints (3.4) and (3.5) examine whether client *i* and client *j* are both able to arrive on time, if client *i* picks up client *j* before going to the destination. Constraint (3.6) checks if the pick-up time of client *j* is too late for client *i* to arrive at the destination on time. Constraint (3.7) guarantees the client *j* lives in the area which can be served by client *i*. If pooling user *i* and *j* together cannot satisfy the above-mentioned constraints, the preference w_{ij} is set to be zero, which means there is no probability that user *i* and *j* are pooled together by ants. By this procedure, we are able to remove some car pool combinations which do not belong to any feasible solution, so the complexity for the ants to search for pool members is significantly decreased.

$$e_i + t_{ij} + t_{j0} \le r_i \tag{3.4}$$

$$e_i + t_{ij} + t_{j0} \le r_j \tag{3.5}$$

$$e_j + t_{j0} \le r_i \tag{3.6}$$

$$t_{ij} + t_{j0} \le T_i \tag{3.7}$$

Then, if all the constraints are well satisfied, the preference values between two different users are initialized by the geographic distance and the time window difference between each two users as (3.8), and the preference values between the users and themselves are computed as (3.9).

$$w_{ij} = \alpha \times \frac{1}{d_{ij}} + \beta \times \frac{1}{|e_i + t_{ij} - e_j| + |r_i - r_j|} \tag{3.8}$$

$$w_{ii} = \theta \times \frac{1}{T_i - t_{i0}} \tag{3.9}$$

where d_{ij} and t_{ij} are the geographical distance and travel time between user i and j, respectively. e_i, e_j, r_i and r_j are the departure-arrival time window of user i and j. T_i is the maximal driving time of user i; t_{i0} is the direct travel time between user i and the destination; α, β and θ are weight factors. Note that the constants in the abovementioned equations have different units. d_{ij} is a distance value in kilometers, while e_i, e_j, r_i, r_j, t_{ij}, t_{i0} and T_i are time values in minutes, so the factors α, β and θ are designed to adjust them, we use an average travel speed 50km/h to transform time values into distances. This method is also used in the future approaches. Furthermore, all the denominators in equation (3.8) and (3.9) are limited to a minimum value σ, which replaces the denominators if their values are less than σ. This mechanism is designed to avoid generating too large preference values.

Equation (3.8) achieves that, while more distance and time window differences between the two users, there is less preference between them. Equation (3.9) calculates a user's tolerance level of having new members inserted into his/her car pool by evaluating the user's extra driving time. If a user has a shorter extra driving time, the user is likely to have less car pool members since his/her service area of picking up group members is smaller.

In another view, the value w_{ii} can be considered as the preference between one user and itself, and it is important for guiding the ants to end the current car pool. When the ant chooses the next user to visit from user i, it also has a probability to still select user i itself as the next location, in this case, the ant will end the current car pool. The purpose of this mechanism is to provide a reference for the ants. By comparing the preference values calculated by equations (3.8) and (3.9), the ants are able to decide it is better to continue searching for new pool members or to stop with the current members. Thus, the ants are given the probability to select a cost-effective opportunity to end the current car pool before reaching the car capacity. For instance, the ant has high probability to end the cluster, when the existing users in current car pool have relatively low preference values to other available users compared with the ones to themselves. This means that all possible users are not economical to be inserted into the current car pool, so the better behavior is to end the cluster rather than pool one more

user into it. This mechanism is essential in CAC as it is a key function to cluster users except the car capacity and time window constraints.

The activities of ants in the CAC are shown in Figure 3.2.

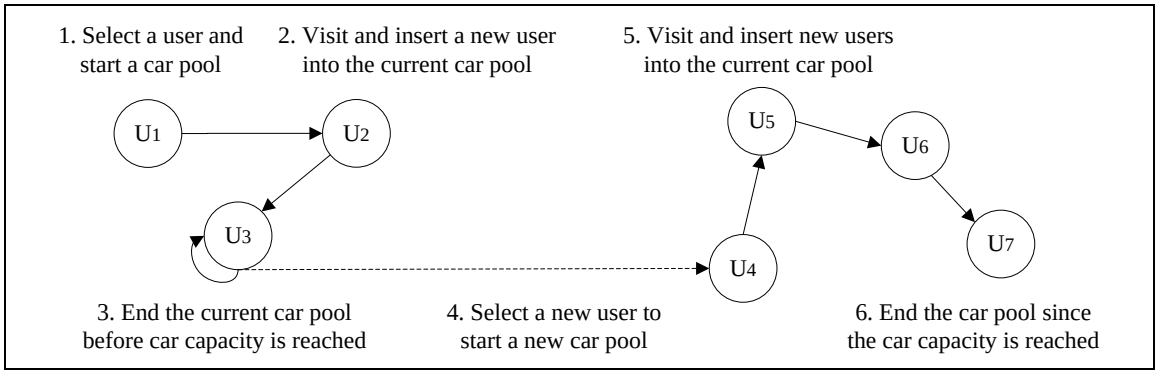


Figure 3.2: Activities of the ants in the CAC.

As abovementioned, the two functions of the preference information, inserting a new user into the current car pool and ending the current car pool, are controlled by a roulette wheel selection procedure. The probability for the ant to select a new user *j* to visit and insert it into the current car pool *k* is based on the preference and attractiveness between user *j* and car pool *k*. In CAC, the preference between a user *j* to a car pool *k* is defined as follows.

Definition 3.4.3

For any user j and car pool k, the preference between them is defined to be the sum of the preference values between user j and each existing member of car pool k, if none of the preference values between user j and each existing member of car pool k equals to zero. Otherwise, the preference between user j and car pool k equals to be zero.

The formulation of definition 3.4.3 is shown in equation (3.10), where *K* is the set of existing members of car pool *k*.

$$\delta_{jk} = \begin{cases} \sum_{i \in K} w_{ij} & if \prod_{i \in K} w_{ij} \neq 0 \\ 0 & otherwise \end{cases} \tag{3.10}$$

A positive preference value gives the ant the probability to insert the user *j* into the car pool *k*. In order to maintain the feasibility after the insertion, all the existing users in the car pool *k* should have a non-zero preference to user *j*, so to prove that at least pooling any of them individually with user *j* is feasible. If any existing user in car pool *k* has a zero preference value to user *j*, which indicates pooling them together violates the time window constraints, the preference between user *j* and car pool *k* is set to be zero. So there is no probability for the ant to insert user *j* into car pool *k*.

However, even all the existing members of car pool *k* have positive preference to user *j*, the insertion may still cause violation of the time window. Thus, the time windows will be examined after each insertion, if there is a time window violation, the insertion will be cancelled and the ant will select another user to start a new car pool.

Therefore, the probability for the ant to select a new user *j* to visit and insert it into the current car pool *k* is calculated as in equation (3.11). In like manner, the probability for the ant to end the current car pool is computed as the sum of preference values of the existing users in car pool *k* to themselves, as shown in (3.12).

$$\mathrm{Ps}_{jk} = \frac{(\delta_{jk})^a \left(\sum_{i\in K} \eta_{ij}\right)^b}{\sum_{m\in H}\left[(\delta_{mk})^a \left(\sum_{i\in K} \eta_{im}\right)^b\right] + \left(\sum_{i\in K} w_{ii}\right)^a \left(\sum_{i\in K} \eta_{ii}\right)^b} \tag{3.11}$$

$$\mathrm{Pe}_{k} = \frac{\left(\sum_{i\in K} w_{ii}\right)^a \left(\sum_{i\in K} \eta_{ii}\right)^b}{\sum_{m\in H}\left[(\delta_{mk})^a \left(\sum_{i\in K} \eta_{im}\right)^b\right] + \left(\sum_{i\in K} w_{ii}\right)^a \left(\sum_{i\in K} \eta_{ii}\right)^b} \tag{3.12}$$

where δ_{jk} and w_{ii} are the preference value between user *j* and car pool *k* and the preference value between user *i* and himself/herself; η_{ij} and η_{ii} are the attractiveness values between user *i*, *j* and the attractiveness value between user *i* and himself/herself, which will be introduced in next section. *K* is the set of existing users in the car pool *k*; *H* is the set of users who have positive preference value with car pool *k*, but haven't been assigned to any car pool; while *a* and *b* are adjusting parameters.

When all ants finish their tours, the first *m* best-fit solutions are selected to be applied a local search, and then these solutions are used to update the preference values. Before updating the preference values with the new solutions, all weight values w_{ij} in the preference matrix will decrease with an evaporate rate μ, in order to enlarge the influence of the new preference information obtained in current iteration. Then for each selected solution *s*, the preference values between the users in the same cluster consist in an update by value φ_s, computed as (3.12).

$$\varphi_s = \lambda \times \frac{f_{avg} - f_s}{f_{avg}} \tag{3.12}$$

where f_{avg} is the average fitness of the whole ant colony; f_s is the fitness of current selected solution *s*. Factor λ is a weight factor used to keep a low value for factor φ_s in anterior iterations and a high value in posterior iterations, so that the ants are more freely to explore the solution space in the beginning iterations.

Thus, the new preference value will be calculated as shown in equation (3.13) where $w_{ij}^{'}$ is the preference value of the previous iteration and S is the set of all selected solutions.

$$w_{ij} = \mu \times w_{ij}^{'} + \sum_{s \in S} \varphi_s \tag{3.13}$$

Figure 3.3 shows an example of preference matrix updating, where user 1, user 2 and user 3 are pooled together in the selected solutions.

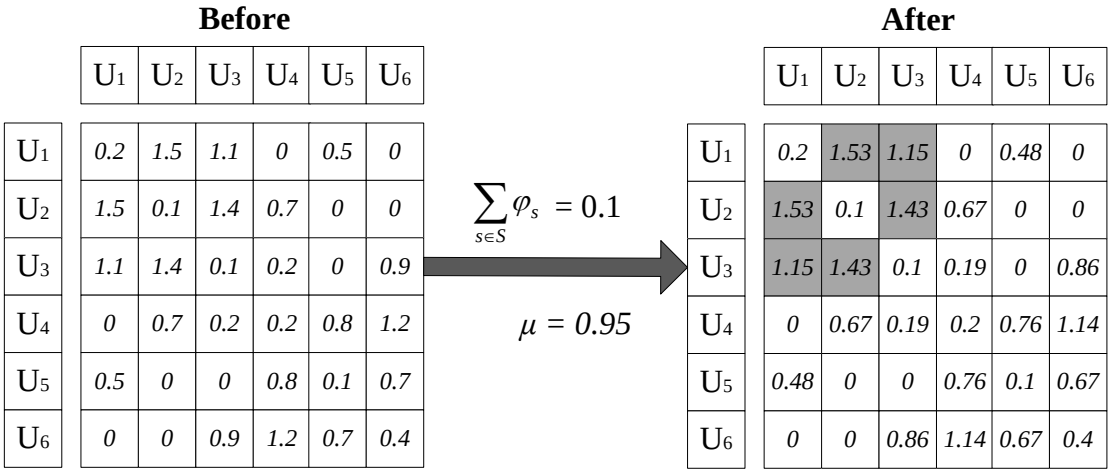


Figure 3.3: Updating the preference matrix

The preference value w_{ii} is also updated with equation (3.14). The update is based on the vacancy level v_i of the vehicles in a car pool. The vacancy level corresponded to user i is calculated as the difference between the minimum car capacity of user i's car pool and the amount of users being pooled in the car pool. For instance, for a car pool with minimum car capacity of four users, if three users have been pooled in the car pool, the vacancy level of each user in the car pool is one. Thus, a higher vacancy indicates the car pool has fewer users compared to its car capacity, which also reveals the users in the car pool have a low tolerance level of having other car pool members. In contrary, if a user has a lower vacancy level, the user is able to accept more other users being pooled together with him. Therefore in equation (3.14), where $w_{ii}^{'}$ is the ancient preference value, if the vacancy level of user i is greater than one in a selected solution s, the preference of user i to itself increases by value φ_s computed in (3.12).

$$w_{ii} = \begin{cases} w_{ii}^{'} & \text{if } v_i \leq 1 \\ w_{ii}^{'} + \varphi_s & \text{otherwise} \end{cases} \tag{3.14}$$

3.4.4 Attractiveness

The basic paradigm of ACO involves the movement of a colony of ants through the different states influenced by two local decision policies, pheromone and attractiveness. In the CAC, the pheromone is replaced by the preference information and the attractiveness η is defined using the equations (3.15) and (3.16). These two equations are the same ones for initializing the preference information; the difference is the attractiveness is computed between every two users, no time window constraint is checked and there is no update for it. The attractiveness can provide the diversity to the algorithm when the preference information converges to an optimum.

$$\eta_{ij} = \alpha \times \frac{1}{d_{ij}} + \beta \times \frac{1}{|e_i + t_{ij} - e_j| + |r_i - r_j|} \tag{3.15}$$

$$\eta_{ii} = \theta \times \frac{1}{T_i - t_{i0}} \tag{3.16}$$

where d_{ij} and t_{ij} are the geographical distance and travel time between user i and j, respectively. e_i, e_j, r_i and r_j are the departure-arrival time window of user i and j. T_i is the maximal driving time of user i; t_{i0} is the direct travel time between user i and the destination; α, β and θ are weight factors. The same limitation value σ as in equation (3.8) and (3.9) are also applied to these two equations.

The attractiveness becomes essential when the ant needs to build a new car pool. After the ant ends the current car pool, it will stochastically search for a new user to start a new car pool, the probability of finding a user is influenced only by the attractiveness. The preference information is ignored in this procedure, since the zero values in the preference matrix will disable the probability to select some users and affect the construction of a complete solution.

So in like manner, the probability for the ant to search for a new user to start a new car pool is computed as the sum of attractiveness between new user j and every existing user in current car pool k, shown in equation (3.17).

$$\mathrm{P}n_{jk} = \frac{\left(\sum_{i \in K} \eta_{ij}\right)^b}{\sum_{m \in H}\left[\left(\sum_{i \in K} \eta_{im}\right)^b\right]} \tag{3.17}$$

where η_{ij} is the attractiveness values between user i and user j; K is the set of existing users in current pool; H is the set of all users who have not been pooled; and b is the adjusting parameter.

3.4.5 Local Search Procedure

After each iteration step, several best solutions are selected to be improved by a local search procedure. The local search operators we used in the CAC are inherited from our VNS-LTCPP approach presented in chapter 2. In order to reduce the complexity of the procedure to gain extra computing speed, the mechanisms of the Divide, Merge and Swap operators are modified, and the Chain operator in VNS-LTCPP is replaced by a new Move operator. The main structure of the local search in CAC consists in a loop considering sequentially each operator, and it stops when no improvement made during *x* iterations. A forbidden list is built to record the unsuccessful attempts of each operator, in order to avoid repetitive runs of unnecessary operations, so the computing resources and time can be further saved. Each operator in the procedure will be applied on several selected car pools, and it moves to the next selected car pool as soon as an improvement is obtained on the current car pool. When all the car pools selected by the current operator are processed, the next operator is applied.

The general structure of the local search procedure is presented in Algorithm 3.3.

***Algorithm 3.3**: Local search in CAC.*

```
Local_Search_Operators[] = { Divide, Merge, Swap, Move }
Do
    For each operator in Local_Search_Operators do
        Select car pools for the operator
        For each selected car pool do
            Check the forbidden list;
                If not in the list, apply the operator;
                    If solution is improved, update the solution;
                    Remove corresponding information of the current car pool from
                    the forbidden list;
                    Else record the operation in the forbidden list;
                Else skip the operation;
         End for
    End for
Until the stop criteria of the local search procedure are met.
Clear the forbidden list.
```

To be more specifically, for each operator, we firstly select one or several car pools according to the selection rules defined by each operator. Then, for selected car pools, we check the forbidden list. If there is corresponding information in the list, which means we applied the same operator to the same car pools in the past and made no

improvement, we will skip the operation. Otherwise, we apply the operator and see whether there is an improvement obtained. If the solution is improved, we update the solution and remove the corresponding information of the modified car pools from the forbidden list. Because the users in the car pools have been changed, the ancient information becomes deprecated and useless. And if the solution is not improved, which indicates the operation on the current selected car pools is not useful. We will record this information in the forbidden list, in order to avoid repeating the same operation again to the same car pools.

Divide operator

The divide operator consists in dividing a selected car pool into smaller car pools with respect to the total travel cost. The operator selects *n%* of the car pools with relatively high intra travel cost; the intra travel cost indicates the total travel cost between each user excluding the costs between the users and the destination. The selection is performed by a roulette wheel selection based on the travel cost of each car pool. The probability of car pool *i* is selected is calculated as equation (3.18).

$$Probability_{pool_i} = \frac{incost_i}{\sum_{j \in K} incost_j} \tag{3.18}$$

where $incost_i$ is the intra travel cost of car pool *i* and *K* is the set of all car pools.

Then, for each selected car pool *i*, the operator tries to pool any members of pool *i* into a new pool. If an improvement is obtained, the move is confirmed. The complexity of this operator is O(n).

Merge operator

The merge operator tries to merge *any* non-full car pools with each other.

For each non-full car pool *i*, other non-full car pools which are able to satisfy the car capacity constraints after merging are put into a list. In this list, the car pools are ordered by their amount of users already in pool; the fewer users in pool, the more frontal in the list. Then the operator tries to merge car pool *i* with each car pool *j* on the list, starting from the top of the list. If an improvement is obtained, the move is confirmed. The complexity of this operator is O(1).

Swap operator

The swap operator tries to swap any two users in two car pools. It first stochastically selects *q% of* car pools which have high intra travel costs; the concept of intra travel cost is the same as in divide operator. The selection is performed by a roulette wheel selection based on the intra travel cost of each car pool. The probability of selecting car pool *i* is also calculated by equation (3.18).

For each selected car pool *i*, the operator selects its nearest car pool *j* according to their gravity centers. Suppose g_{ij} is the distance between the gravity centers of car pool *i* and car pool *j*, then the smallest g_{ij} is selected to locate the car pool *j* for each car pool *i*. Then, it tries to swap every member in pool *i* with every member in pool *j*. If an improvement is obtained, the move is confirmed. The complexity of this operator is $O(n^2)$.

Move operator

The move operator tries to move a user from a selected car pool into a non-full car pool. It first stochastically selects *k% of the* pools. The selection is performed by a roulette wheel selection based on the intra travel cost of each car pool. Equation (3.18) is still used to calculate the probability of selecting car pool *i*.

For each selected pool *i*, the operator selects its nearest non-full car pool *j* according to the gravity centers. In the like manner of swap operator, suppose g_{ij} is the distance between the gravity centers of car pool *i* and car pool *j*, then the smallest g_{ij} is selected to locate the car pool *j* for each car pool *i*. Then, the operator tries to move every member of pool *i* into pool *j*, one member for each attempt. If an improvement is obtained, the move is confirmed. The complexity of this operator is O(n).

Forbidden List

The unsuccessful attempts of each operator are recorded in this forbidden list, shown in Figure 3.4.

Target pool 1	Target pool 2	Operator
12	—	Divide
3	4	Swap
17	22	Move
…	…	…

Figure 3.4: An example of the forbidden list.

For instance, in the second row of figure 3.4, the Swap operator tried to swap each user in car pool *3* and in car pool *4*, and the evaluation of this operation shows no improvement of the sum of the cost of the two car pools, then this operation is considered as an unsuccessful attempt and will be recorded into the Forbidden list. The complexity of insert and remove operation made on this list is *O(n)*.

If a car pool is modified by any operator, in other words, the members in the car pool have been changed. Then the records in forbidden list concerning this car pool will be removed. Otherwise, if any operator tries to apply the same operation stored in the forbidden list to the same car pool or car pools, the operation will be skipped. The forbidden list will be cleared when the stop criteria of local search are met. This mechanism can significantly decrease the operation time of the local search procedure, since lots of unsuccessful attempts only have to be applied once. Based on our

experiments, the forbidden list can save 20% to 50 % of the computing time of the local search procedure.

3.5 Computational Results

Computational experiments have been conducted to compare the performance of the proposed approach with other existing metaheuristics for solving the LTCPP.

3.5.1 Configuration

Parameter setting for the investigated algorithm is specified in table 3.1. Given limited computational resources and combinatorial complexity, parameter values were determined empirically over a few intuitively selected combinations, choosing the one that yielded the best average output. There are 14 parameters for the CAC approach.

Number of Ant	Initialization				Probability		Preference			Local search			
	α	β	σ	θ	a	b	m	λ(iteration)	μ	n	q	k	x
90	0.9	0.1	1	5	2	1	10	0.1(<300) 0.5(≥300)	0.95	10	20	20	2

Table 3.1: Parameter setting for the Clustering Ant Colony Algorithm.

3.5.2 Experimental Results

In order to provide an exclusive evaluation of the performance of our algorithm, our experimental results are compared with three other approaches for solving the LTCPP, the ANTS and the simulation-based approach (SB), as well as the VNS-LTCPP presented in chapter 2. The reason for selecting ANTS approach to compare with our approach is that, both ANTS and our approach are based on the ant colony structure, but very different in defining the pheromone and attractiveness concepts. Thus, we believe the ANTS could be a valuable reference to evaluate the performance of our approach. Since the benchmarks and implementation environments of the ANTS is different from the other approaches, in order to provide a fair and convincing comparison, we implemented both approaches in the same environment with JAVA and use them to solve the benchmarks presented previously. Please note the ANTS approach is implemented exactly as it is described in the literature with no modification.

The experiments consist in performing 30 simulation runs for each problem instance on Windows operating system with Intel Core i7 740QM 2.9 GHz CPU and 4 GB RAM. The CAC are given 1000 iterations and then the ANTS generates the

same amount of solutions as the CAC, while the SB and VNS-LTCPP approaches are set to run until no improvements can be found.

Table 3.2 compares the experimental results of the C set instances. The CAC outperforms other approaches on 7 instances considering the average solution quality of 30 runs. Furthermore, the CAC's best found solutions of all the instances are better than or equal to the ones provided by other approaches.

Inst	Size	CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]			VNS-LTCPP		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
C101	100	1585.5	1593.4	11	1585.5	1592.9	17	1647.4	1669.2	91	1644.9	1684.6	7
C102	100	1706.8	1728.2	10	1711.4	1748.5	14	1717.5	1724.8	94	1729.2	1753.8	9
C103	100	1508.6	1527.5	11	1512.6	1535.1	18	1532.2	1599.4	85	1545.9	1563.6	7
C201	200	2703.1	2717.7	25	2784.4	2854.2	57	2761.7	2868.6	329	2676.9	2723.8	24
C202	200	2879.2	2892.9	36	2936.1	3004.5	64	3081.7	3114.1	473	3070.7	3145.2	26
C203	200	2769.3	2834.1	29	2845.9	3003.5	58	2975.1	3182.4	394	2935.1	2993.7	37
C401	400	5533.3	5618.6	189	5833.5	6281.4	424	6174.2	6860.3	934	6024.5	6130.1	255
C402	400	4518.2	4760.3	242	4893.5	5153.2	357	5383.7	5524.5	683	5055.9	5110.7	197
C403	400	5930.7	6046.4	271	6125.6	6742.1	511	6675.2	6994.5	1257	6079.6	6322.5	287
Total		29134.7	29719.1	824	30228.5	31915.4	1520	31948.7	33537.8	4340	30762.7	31428.0	849

Table 3.2: Experimental results of set C instances (clustered user distribution).

Table 3.3 shows the percentage the CAC outperforms other approaches on set C instances, in the aspects of average solution quality and computing time. For each instance, the outperforming percentage is calculated as *(other approach's value – CAC's value) / other approach's value*. Each value in table 3.3 is obtained by averaging the outperforming percentages of the three same-size instances. Comparing with ANTS and SB, the CAC can provide better solution quality in much less computing time. The VNS-LTCPP's solution quality has been improved 4.7% by the CAC with similar computing time (considering the summation of computing time in table 3.2).

Set	Size	ANTS [Maniezzo, 2004]		SB [Correia, 2007]		VNS-LTCPP	
		Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
C	100	0.54%	34.25%	2.95%	88.11%	3.06%	-41.80%
	200	4.71%	49.96%	7.77%	92.48%	4.53%	-7.00%
	400	9.50%	44.87%	15.16%	74.26%	6.52%	2.87%
Avg		4.92%	43.03%	8.63%	84.95%	4.70%	-15.31%

Table 3.3: Solution quality and computing time comparison on set C instances.

Table 3.4 presents the experimental results on the R set instances. The CAC shows some difficulties in solving the random distributed instances. It outperforms other approaches on only 4 instances in best found solution, and on 6 instances considering the average solution quality. Considering the total value in the bottom row, the CAC still provides better solution quality, and the computing time is still significantly less than other approaches. But the ANTS and the SB approach provided the best found solutions for most instances with 100 and 200 users, which indicates the CAC is not very competitive in solving small size instances with random distributed users. Table 3.5 reveals the percentage of outperforming in the same manner as table 3.3.

Ints	Size	CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]			VNS-LTCPP		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
R101	100a	2223.1	2283.2	12	2207.1	2281.5	18	2235.1	2265.4	100	2211.2	2286.6	8
R102	100b	1841.4	1874.3	13	1834.6	1864.2	17	1832.8	2091.7	97	1856.7	1898.7	10
R103	100c	2235.9	2313.4	11	2299.2	2438.7	21	2204.7	2418.5	80	2288.3	2379.8	7
R201	200a	4156.1	4231.8	36	4101.5	4253.5	108	4425.0	4567.1	430	4349.2	4464.6	35
R202	200b	3717.2	3824.2	29	3772.2	4071.9	84	3952.4	4283.3	231	3970.3	4162.0	27
R203	200c	4164.7	4304.6	44	4368.5	4541.5	116	4092.4	4257.5	540	4118.5	4282.5	36
R401	400a	7891.4	8033.7	358	8396.1	8580.4	581	8787.8	8993.8	1106	8097.1	8398.4	316
R402	400b	6365.2	6559.7	304	6512.7	6893.3	479	7258.7	7417.5	896	6411.8	6975.0	196
R403	400c	8023.4	8129.5	289	8113.1	8338.9	631	8841.9	8933.5	1037	8312.1	8422.0	309
Total		40618.4	41554.4	996	42505	44663.9	2055	43630.8	45228.3	4517	41615.2	43269.6	944

Table 3.4: Experimental results of set R instances (random user distribution).

Size	ANTS [Maniezzo, 2004]		SB [Correia, 2007]		VNS-LTCPP	
	Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
100	1.51%	34.83%	4.65%	86.95%	1.41%	-45.71%
200	3.94%	64.74%	5.65%	90.31%	4.27%	-10.83%
400	4.57%	50.00%	10.41%	72.33%	4.59%	-3.63%
Avg	3.34%	49.85%	6.91%	83.20%	3.42%	-20.06%

Table 3.5: Solution quality and computing time comparison on set R instances.

The experimental results for the set W instances (real-world instances) are presented in table 3.6. The CAC provides better results on 8 instances both in best found solution quality and average solution quality. The outperforming percentage is shown in table 3.7.

Inst	Size	CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]			VNS-LTCPP		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
W101	100	864.2	886.3	9	889.4	893.8	18	864.2	885.7	81	866.7	886.9	7
W102	100	998.9	1008.5	11	1016.5	1020.3	18	1007.5	1037.9	82	1007	1041.5	8
W103	100	1112.4	1134.8	10	1142.4	1168.1	20	1100.5	1187.6	87	1118.9	1173.8	8
W201	200	1523.1	1557.6	46	1586.1	1601.8	97	1717.1	1722.9	341	1614.4	1682.5	38
W202	200	1803.7	1812.9	22	1872.2	1919.2	104	2014.3	2127.7	406	1943.5	2003.6	25
W203	200	1701.2	1784.4	40	1795.6	1907.5	85	1860.5	1965.1	371	1733.7	1806.8	44
W401	400	2789.5	2848.4	356	2843.4	3066.4	481	3168.4	3442.8	955	2975.2	3076.8	520
W402	400	3283.1	3360.1	324	3541.6	3692.9	441	3633.5	3984.9	792	3625.6	3713	306
W501	565	4887.5	5056.2	515	5090.2	5224.9	697	5377.5	5858.3	1621	5105.2	5288.4	578
Total		18963.6	19449.2	1333	19777.4	20494.9	1961	20743.5	22212.9	4736	19990.2	20673.3	1534

Table 3.6: Experimental results of set W instances (real world cases).

Size	ANTS [Maniezzo, 2004]		SB [Correia, 2007]		VNS-LTCPP	
	Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
100	1.62%	46.30%	2.40%	87.99%	2.19%	-30.36%
200	4.92%	61.45%	11.20%	90.10%	6.06%	0.01%
400	6.45%	26.21%	15.55%	63.35%	7.11%	12.19%
Avg	4.33%	46.45%	9.71%	80.48%	5.12%	-6.05%

Table 3.7: Solution quality and computing time comparison on set W instances.

Considering all three sets of instances, the solution quality of CAC is significant better than other approaches when dealing with the instances with 200 and 400 users in all three sets. However, the search ability on the instances of random distributed users needs to be improved.

A further evaluation of the performance among the four approaches has been conducted by using a Friedman test [Friedman, 1940]. The test consists in the average solution quality on all 27 instances used in our experiments. The detail information is presented in table 3.8, the detail calculation can be found in appendix 2.

The Friedman statistic value T of table 3.8 is calculated to be *24.63*, while the threshold for the F distribution with a significance level 0.01 is *4.04*. Since T is much greater than the threshold, it is proven that there exists at least one approach whose performance is significant different from at least one of the other approach. A paired comparison is then performed to decide which approaches are really different; the detail calculation is also presented in appendix 2. According to the paired comparison, for significance level 0.01 and 78 degrees of freedom, the critical value for a significant difference between two approaches is *18.29*. Table 3.9 shows the difference between the performances of every two approaches. We can see that our CAC approach provides the best performance. It outperforms significantly all the other approaches.

Instance	CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]			VNS-LTCPP		
	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2
C101	1593.4	2	4	1592.9	1	1	1669.2	3	9	1684.6	4	16
C102	1728.2	2	4	1748.5	3	9	1724.8	1	1	1753.8	4	16
C103	1527.5	1	1	1535.1	2	4	1599.4	4	16	1563.6	3	9
C	2717.7	1	1	2854.2	3	9	2868.6	4	16	2723.8	2	4
C202	2892.9	1	1	3004.5	2	4	3114.1	3	9	3145.2	4	16
C203	2834.1	1	1	3003.5	3	9	3182.4	4	16	2993.7	2	4
C401	5618.6	1	1	6281.4	3	9	6860.3	4	16	6130.1	2	4
C402	4760.3	1	1	5153.2	3	9	5524.5	4	16	5110.7	2	4
C403	6046.4	1	1	6742.1	3	9	6994.5	4	16	6322.5	2	4
R101	2283.2	3	9	2281.5	2	4	2265.4	1	1	2286.6	4	16
R102	1874.3	2	4	1864.2	1	1	2091.7	4	16	1898.7	3	9
R103	2313.4	1	1	2438.7	4	16	2418.5	3	9	2379.8	2	4
R201	4231.8	1	1	4253.5	2	4	4567.1	4	16	4464.6	3	9
R202	3824.2	1	1	4071.9	2	4	4283.3	4	16	4162	3	9
R203	4304.6	3	9	4541.5	4	16	4257.5	1	1	4282.5	2	4
R401	8033.7	1	1	8580.4	3	9	8993.8	4	16	8398.4	2	4
R402	6559.7	1	1	6893.3	2	4	7417.5	4	16	6975	3	9
R403	8129.5	1	1	8338.9	2	4	8933.5	4	16	8422	3	9
W101	886.3	2	4	893.8	4	16	885.7	1	1	886.9	3	9
W102	1008.5	1	1	1020.3	2	4	1037.9	3	9	1041.5	4	16
W103	1134.8	1	1	1168.1	2	4	1187.6	4	16	1173.8	3	9
W201	1557.6	1	1	1601.8	2	4	1722.9	4	16	1682.5	3	9
W202	1812.9	1	1	1919.2	2	4	2127.7	4	16	2003.6	3	9
W203	1784.4	1	1	1907.5	4	16	1965.1	3	9	1806.8	2	4
W401	2848.4	1	1	3066.4	2	4	3442.8	4	16	3076.8	3	9
W402	3360.1	1	1	3692.9	3	9	3984.9	4	16	3713	2	4
W501	5056.2	1	1	5224.9	2	4	5858.3	4	16	5288.4	3	9
Avg		1.30			2.41			3.41			2.89	
Sum		35	55		65	173		92	344		78	238

Table 3.8: Friedman test results.

$\lvert R_i-R_j \rvert$	ANTS [Maniezzo, 2004]	SB [Correia, 2007]	VNS-LTCPP
CAC	30	57	43
ANTS [Maniezzo, 2004]	-	27	13
SB [Correia, 2007]	-	-	14

Table 3.9: Paired comparison results.

Another experiment has been conducted to evaluate the effect of the local search procedure. In the experiment, the local search procedure is removed from the CAC to generate a pure CAC approach (PCAC). The approach is given the same computing time as the CAC in the previous experiments. Table 3.10 shows the *average* of their solution qualities of the three same-size instances. The average solution quality differences between the CAC and PCAC of the three sets of instances are 4.86%,

7.38% and 4.17%, respectively, which indicates the local search procedure makes its effort in improving the CAC's solution quality.

Size	C set instances			R set instances			W set instances		
	CAC	PCAC	Diff(%)	CAC	PCAC	Diff(%)	CAC	PCAC	Diff(%)
100	1609.7	1648.8	2.43	2057	2127.4	3.42	1009.9	1023.1	1.31
200	2838.2	2972.4	4.73	4006.9	4317.4	7.75	1718.3	1780.2	3.60
400	5475.1	5881.8	7.43	7541	8369.4	10.98	3754.9	4039.8	7.59
Avg	3307.7	3501.0	4.86	4535.0	4938.0	7.38	2161.0	2281.0	4.17

Table 3.10: Evaluation of the effect of the local search procedure.

The accuracy of the CAC is examined by calculating the standard error (column Std) of the solutions obtained in 30 runs of each instance. The solution quality difference (column Diff) between the best found solution quality and the average solution quality of *each* instance in the previous tables is also calculated. Table 3.11 shows the *average* of the abovementioned values of the three same-size instances. The average differences between the best found solution and the average solution of the three sets of instances are 1.3%, 1.6% and 2.3%, respectively, which indicates the CAC approach can be considered to be accurate for a metaheuristic.

Size	C set instances				R set instances				W set instances			
	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)
100	1601.6	1609.7	4.12	0.5	2033.5	2057.0	13.28	1.1	991.8	1009.9	9.08	1.8
200	2797.2	2838.2	24.82	1.4	3942.7	4006.9	41.26	1.6	1676.0	1718.3	21.54	2.5
400	5374.1	5475.1	65.66	1.8	7390.0	7541.0	92.59	2.0	3653.4	3754.9	49.32	2.7
Avg	3257.6	3307.7	31.53	1.3	4455.4	4534.9	49.04	1.6	2107.1	2161	20.40	2.3

Table 3.11: Evaluation of the accuracy of the CAC.

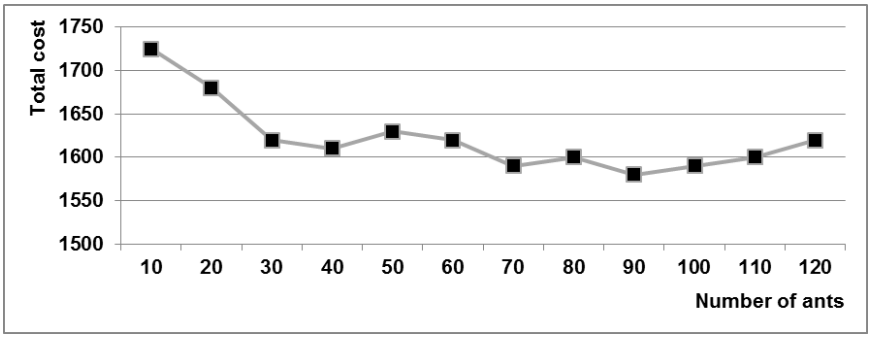


Figure 3.5: Evaluation of the stability of the CAC by modifying the number of ants.

The stability of CAC is evaluated by modifying the number of ants but maintaining same amount of solutions generated. Figure 3.5 shows the solution quality change when modifying the number of ants. From the figure, it is easy to notice that the

solution quality stays stable during the change of the number of ants. Most of the best average solution quality is obtained with 90 ants, therefore this amount is chosen for the experimentations.

3.6 Conclusion

In this chapter we introduced CAC, a new clustering ant colony algorithm to solve the LTCPP. We defined a preference concept into the ACO system to replace the traditional pheromone information, which converts the classic ACO into a methodology with both clustering and routing abilities. In our approach, the preference information is used to guide the movement direction and the car pool construction behavior of the ants. Then, a local search is defined to further optimize the best solutions obtained during iteration.

The presented approach has been applied successfully for solving the long-term car pooling problem. Experiments have been performed to confirm the efficiency and the effectiveness of the preference mechanism. The CAC approach has been also compared with other existing metaheuristics for solving the long-term car pooling problem. For the instances with clustered user distribution and the real-world instances, the CAC can provide significant better solution quality compared with other metaheuristics. However, the performance of the CAC on random distributed instances is less preferable. The reason the CAC's performance decreases on the instances with random distributed users is that the resolution ability of the ACO framework relatively depends on the distribution of the users. The clustered distributed users are easier for the ants to construct good quality clusters, since the differences between a user's high preference values and the low ones are significant, which provide a strong guidance for the ants. In the other hand, the preferences among the random distributed users are relatively similar, which increases the difficulty for the ants to find high quality combinations of users. Thus, we believe the solution quality of the instances with random distributed users can be further improved by other metaheuristics, which drives us to further explore other population based metaheuristics.

The mechanism of the genetic algorithm, which is based on applying recombination and mutation operations on the representation of the solution, is affected less by the distribution of the users. So we believe it is a good candidate for solving the LTCPP. In the next chapter, we will discuss our genetic algorithm based approaches.

CHAPTER 4

Evolutionary Metaheuristics for the Long-term Car Pooling Problem

CONTENTS

Abstract

In this chapter, a Guided Genetic Algorithm and a Multi-agent Self-adaptive Genetic Algorithm are introduced. The presented approaches have been applied successfully for solving the LTCPP. Experiments have been performed to confirm the efficiency and the effectiveness of the algorithms.

4.1 Introduction

The evolutionary metaheuristics belong to the population-based metaheuristic family. The underlying idea is that, given a population of individuals, the environmental pressure causes natural selection which results a rise in the fitness of the population. It typically uses some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Candidate solutions of an optimization problem play the role of individuals of the population, and the fitness function measures the solutions. Some of the better individuals are chosen to seed the next generation by applying recombination and mutation to them. Recombination is an operation applied to two or more selected individuals and results one or more new individuals. Mutation is applied to one individual and results in one new individual. Executing recombination and mutation leads to a set of new individuals, which also called offspring. The offspring and their parents are selected based on the fitness for a place in the next generation. Evolution of the population then takes place after the repeated application of the above operations.

In the evolutionary metaheuristic family, we select the Genetic Algorithm (GA) as the basic paradigm to solve the LTCPP, since GA is the most selected for solving the optimization problems and its solution representation is most suitable for the LTCPP. The optimization mechanism of the GA, which is based on applying recombination and mutation operations on the representation of the solution, is affected very less by the distribution of the users, thus we believe it is a good candidate for solving the LTCPP.

In recent years, the GA has become to be the most popular evolutionary metaheuristic. The algorithm appears to be particularly appropriate for solving the routing and scheduling problems. It is often applied as an approach to find global optimization solutions. The GA is proven to be useful in problem domains that have a complex mixed fitness landscape. The combination of recombination and mutation operations is designed to move the population away from local optima that other algorithms might get stuck in. Based on the above mentioned reasons, we present firstly in this chapter a Guided Genetic Algorithm (GGA) for solving the Long-term Car Pooling Problem. In the Guided Genetic Algorithm, the composition of the better individuals will always be memorized and updated. Then this information will be used for aiding the genetic operators, in order to produce more feasible offspring solutions with high

solution quality. Furthermore, the GGA is designed with an adaptive control of the variation rates. The algorithm adapts the recombination rate and mutation rate dynamically, in order to maintain stable population diversity at a desirable level.

However, some weaknesses with population based metaheuristics for the long-term car pooling problem appears during the research. First, although the use of metaheuristics allows to significantly reducing the computational complexity of the search process, the latter remains time or memory consuming for the large size instances. Second, the diversity of the population decreases significantly after the convergence to an optimum. Third, the population-based algorithms require a large number of accurate parameter settings in order to obtain good search ability. At last, the structures of the algorithms are always fixed, thus the new operators or constraints are hard to insert into or remove from the system without modifying the algorithm structure. Therefore, we are motivated to develop an improved approach for the LTCPP. This can be achieved by a multi-agent system with hyper-heuristic. The multi-agent system (MAS) is able to improve the computational speed and maintain the diversity after convergence by communicating among the agents. The hyper-heuristic is used to find the most suitable operator or sequence of operators, thus the design of each individual operator becomes more flexible. Furthermore, with the hyper-heuristic, any new operator can be easily inserted into the system without modifying the system's main structure, since the hyper-heuristic will select the most appropriate operator to apply.

Thus, we investigate then in this chapter to merging the GA with the multi-agent system and the hyper-heuristic. For this purpose, we elaborate a Multi-agent Self-adaptive Genetic Algorithm for solving the long-term car pooling problem.

This chapter starts with an introduction to classic genetic algorithm and the common concepts related to it. In Section 4.3 the Guided Genetic Algorithm is introduced. The experimental results of the GGA approach are presented in section 4.4. Section 4.5 presents our Multi-agent Self-adaptive Genetic Algorithm for solving the LTCPP, while the performance of the AGA algorithm is assessed and discussed in section 4.6. Finally, in section 4.7, we conclude this chapter with a summary and introduce the future step of this thesis.

4.2 Classic Genetic Algorithm

There exists various approaches in the evolutionary metaheuristic family, but the difference among them is not very significant. The algorithms mainly differ in the representation of the solution. Typically, the solutions are represented by string over a finite alphabet in Genetic Algorithms (GA) [Fraser and Burnell, 1970], real-valued vectors in Evolution Strategies (ES) [Schwefel, 1981], finite state machines in Evolutionary Programming (EP) [Fogel et al., 1966] and trees in Genetic Programming (GP) [Cramer, 1985]. These differences have a mainly historical origin. An approach of the

evolutionary metaheuristic family is selected for solving a problem is usually because it matches the given problem better, that is, it makes the encoding of solutions easier or more natural. For solving the LTCPP, the straightforward choice is to use the string representation, hence the most suitable evolutionary metaheuristic is the Genetic Algorithm.

Because of the similarity among the approaches in the evolutionary metaheuristics, we only present the classic GA as the representative of the whole family in this section.

The principles of classic GA [Fraser and Burnell, 1970; Goldberg, 1989] are well known, it follows the typical procedure of the evolutionary metaheuristic. In GA, a population of strings, usually called chromosomes, which encodes candidate solutions, normally called individuals, to an optimization problem, evolves toward better solutions. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated; multiple individuals are stochastically selected from the current population based on their fitness. Then, the selected individuals are recombined and mutated to form offspring in order to generate a new population. The offspring exhibit some of the characteristics of each parent, and the new population is then used in the next iteration of the algorithm. Analogous to the biological processes, offspring with relatively good fitness levels are more likely to survive and reproduce, with the expectation that fitness levels throughout the population will improve as it evolves. Figure 4.1 shows the operations of a classic GA. Commonly, the algorithm terminates when either a maximum number of generations have been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. The structure of the GA is presented in Algorithm 4.1.

Algorithm 4.1*: Classic Genetic Algorithm.*

```
Initialize();    /* Generate initial population */
Evaluate();    /* Evaluate each individual in the population */

While not Termination Criterion () do
    Selection();    /* Select the parent individuals */
    Recombination();    /* Recombine the parent individuals */
    Mutation();    /* Mutate some of the individuals */
    Evaluation();    /* Evaluate each individual in the population */
    Survival();    /* Update the population for the next generation */
End while
```

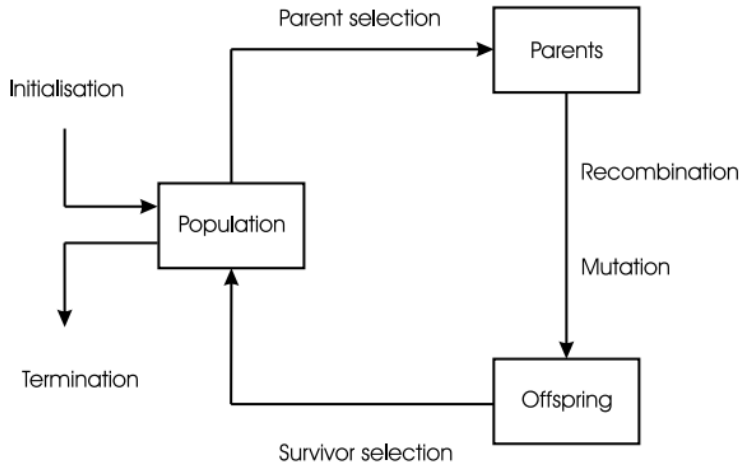


Figure 4.1: The general scheme of a genetic algorithm.

There are a number of components that must be specified in order to define a particular Genetic Algorithm. They are introduced as follows.

4.2.1 Representation

The representation is the connection between the problem context and the problem solving space where evolution will take place. Both direct coding and indirect coding are commonly used in the solution representation of GA, but normally a standard representation is a string. The main reason to use string representations is that their parts are easily aligned due to their fixed size, which facilitates simple recombination operations.

4.2.2 Fitness Function

The fitness function is defined over the genetic representation and measures the quality of the represented solution. It forms the basis for selection, and thereby it facilitates improvements. The fitness function is always problem dependent. If the original problem to be solved by a GA is an optimization problem, the fitness function can be identical to, or a simple transformation of the objective function of the given problem.

4.2.3 Population

The role of the population is to hold possible solutions. A population is a set of individuals, and it forms the evolution. Defining a population is simple; the only necessary parameter to set is the size of the population.

As opposed to the recombination or mutation operators that act on one or two in-

dividuals, the selection operators, which include both parent selection and survival selection, work at the population level. In GA, the population size is constant, not changing during the evolutionary search.

A very important concept to note is the diversity of a population. It indicates the number of different individuals in the population. A well-selected population should always maintain its diversity during the evolution.

4.2.4 Selection

There a two selection operations in the classic GA, the parent selection and the survival selection.

The role of parent selection is to distinguish among individuals based on their quality, in particular, to allow the better individuals to become parents of the next generation.

Definition 4.2.1
An individual is a parent if it has been selected to undergo variation in order to create new individuals, while the created individuals are considered as the offspring of the corresponding parents.

Together with the survival selection, parent selection is responsible for pushing the quality improvement of the population. In GA, parent selection is typically probabilistic, thus high quality individuals get a higher probability to become parents than those with low quality. Nevertheless, low quality individuals are often given a small, but positive chance, in order to avoid the algorithm getting stuck in a local optimum.

The survival selection is to distinguish among individuals based on their quality. It is similar to parent selection, but it is used in a different stage of the evolutionary cycle. The survival selection is applied after creating the offspring of the selected parents. As previous mentioned the population size of a GA is constant, thus a choice has to be made on which individuals will be allowed in the next generation. This decision is usually made also based on the fitness value, favoring those with higher quality.

4.2.5 Variation operators

The variation operators aim to create new individuals from old ones. They are divided into two types: recombination and mutation.

The recombination operator is also called crossover operator. It merges characteristics of two parents into one or two offspring. The choice of what parts of each parent are combined is normally stochastic. The main principal of recombination is simple. By mating two parents with different desirable features, we can produce an offspring

which combines both of those features, which may provide a higher fitness quality.

The mutation operator is applied to one individual and delivers a slightly modified mutant. The original idea to apply this operator is to connect the search space, that is, the mutation operator can move to anywhere during the evolution. However, the randomness of the move possesses limited practical importance. Hence, many implementations of GA use a guided mutation procedure.

These variation operators ultimately result that, in the next generation of population, the individuals are different from the ones in the current generation. Generally the average fitness will have increased by this procedure for the population, since the individuals with higher quality have a higher probability to be selected for breeding.

It is important to note that variation operators are representation dependent. That is, for different representations different variation operators have to be defined.

4.2.6 Initialization

Initialization is kept simple in most GA applications. Typically, initial individual solutions are randomly generated to form an initial population, allowing the entire range of possible solutions. In practice, problem specific heuristics can be used aiming at an initial population with higher fitness, however the search space may be limited and extra computational time is required. Whether this is worth the extra computational effort or not is very much depending on the optimization problem in hand.

4.3 Guided Genetic Algorithm for the LTCPP

As above mentioned, in classic GA, the offspring solutions are produced which exhibit some of the characteristics of each parent. However with a classic recombination operator, the process always generates a large amount of solutions with low solution quality. A proper amount of low quality solutions may enlarge the search of the solution space, but too many of them could decrease the probability to find a good solution close to the optimum. To improve the classic GA in solving the LTCPP, we adapt the preference mechanism from the CAC algorithm presented in chapter 3 into the genetic paradigm. In our Guided Genetic Algorithm (GGA), the composition of the better individuals will always be memorized and updated by the preference information. Then this information will be used for aiding the recombination and mutation operators, in order to produce more offspring with high fitness. According to our experiments, with this mechanism, fitness quality of the offspring is significantly increased. Moreover, we propose an adaptive setting of the recombination and mutation rates in order to control the intensity and the diversity of our algorithm at a desirable level, so the algorithm can progress on the search space but does not converge easily.

4.3.1 Main Structure of the GGA

The overall structure of the GGA is presented in algorithm 4.2. Our approach follows the paradigm of the classic GA as introduced in the previous sections. The initial population is generated in the beginning of the algorithm. Then, the information for guiding the variation operators is also initialized, and the initial recombination and mutation rates are set. In the next step, the algorithm creates new population by repeating the following procedures. First, pairs of parents are selected from the population. Second, with the previous defined rates, the variation operators proceed with the recombination and mutation on the selected parents to breed offspring. Third, the new offspring are evaluated, and the compositions of better offspring are recorded by updating the preference information. At last, the survival selection is applied, and the individuals with better fitness will survive to form the new population. Then, the recombination and mutation rates are adjusted according to the diversity level of the new population.

Algorithm 4.2*: Guided Genetic Algorithm.*

```
Generate initial population;
Initialize the guidance information;
Set initial recombination and mutation rates;
Evaluate();    /* Evaluate each individual in the population */

While not Termination Criterion do
    While not exceed the recombination rate do
        Selection();    /* Select the parent individuals */
        Recombination();    /* Recombine the parent individuals */
        Mutation();    /* Mutate the offspring according to mutation rate */
    End while
    Evaluation();    /* Evaluate the generated offspring */
    Update the guidance information;
    Survival();    /* Update the population for the next generation */
    Adjust the recombination and mutation rate;
End while
```

4.3.2 LTCPP Solution Representation

The representation of a classic GA refers to encoding solutions in the form of a string. Individual positions within each chromosome are referred to as genes. In GAs, both

direct and indirect representations have obtained many successful implementations. Based on the structure of the solutions of LTCPP, we choose direct coding representation as our strategy. Same as the solution representations of the approaches presented in the previous chapters, the solution representation of the GGA is also designed with two levels. The upper level is the chromosome representation, which will be processed by the variation operators. It is encoded as a serial of groups where each group represents a car pool, and each group is expressed by sequential integers where each integer indicates a user in this car pool. In the second level, some other information is collected and associated with each user, such as the total travel time, total travel distance, and the time schedule to pick up the car pool members when this user acts as a server.

4.3.3 Selection

Following the same procedure of the classic GA, two selection operations are conducted in the GGA: parent selection and survival selection.

During each successive generation, a proportion of the existing population is selected to be parents in order to breed offspring. The amount of selected population corresponds to the recombination rate. To perform selection, both roulette wheel selection and tournament selection mechanism [Miller and Goldberg, 1995] were tested during the design of the algorithm. The latter is finally chosen because its selection pressure is easily adjusted. Tournament selection involves running several tournaments among a few individuals chosen at random from the population. The individual with the best fitness of each tournament has greater opportunity to be selected for recombination. Selection pressure is adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected. In GGA, two individuals are selected from the population as candidate solutions by the binary tournament method. Thus, they are chosen from the population at random. The one with the better fitness value is chosen to be the first parent. The process is repeated to obtain a second parent. The objective function of the LTCPP presented in chapter 1 is used to evaluate the fitness of each individual.

After new offspring are generated by the variation operators, they are put together with the ancient population to face a survival selection. The first 10% solution with the best fitness value always survives, and a binary tournament selection is performed to select the rest individual that can survive to the next generation. Similar to the previous parent selection, two individuals are selected randomly, and the one with better fitness is put into the new population and removed from the selection pool. The process repeats until the new population obtains enough individuals.

The selection operations in GGA are responsible for pushing the quality improvement of the population, which intensifies the search procedure.

4.3.4 Recombination

Reproduction is one of the most crucial functions in the chain of genetic evolution. It serves the important purpose of combining the useful traits from parent individual and passing them onto the offspring. The recombination rate decides the percentage of the population that will be selected to breed the offspring, and it is adaptive to the diversity of the population in GGA.

The two selected parent individuals are recombined by being applied with a recombination operator. In the representation of LTCPP, where each integer element appears only once in the chromosome, we decide to use the 2-point recombination, in which two points in the chromosome are chosen randomly, but each point must be a start or an end of a car pool. One offspring consists of the genes from the first parent which are between the two chosen points in the chromosome, along with the genes from the second parent which are to the left of the first point and to the right of the second point of the chromosome. After these two parts of genes have been selected, the duplicate users caused by the recombination will be removed from the second part; and the users which do not exist in any part of genes will be randomly selected and inserted in to the chromosome based on the preference information. New car pools can be created to prevent invalid offspring from being reproduced. A second offspring is produced by swapping round the parents then using the same procedure. The mechanism of the recombination operator is graphically illustrated in figure 4.2.

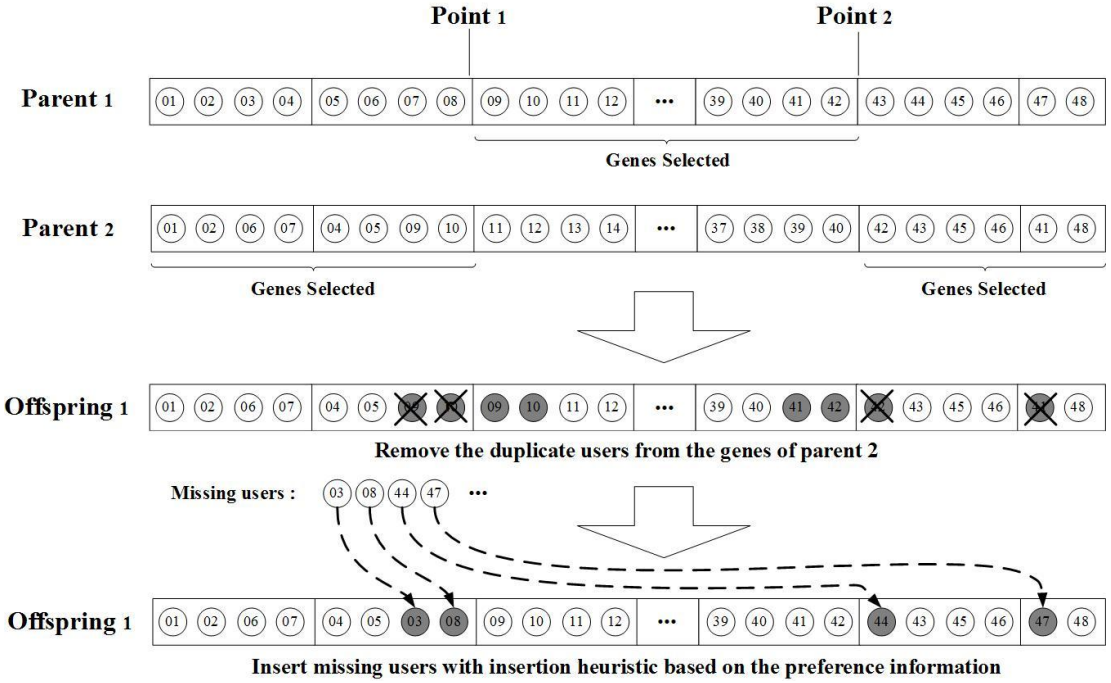


Figure 4.2: Mechanism of the recombination operator.

In genetic process, the variation operators will modify the parent population in order to produce the offspring. During this procedure, the chromosome will be decomposed and then recombined.

In the context of the LTCPP, a classic 2-point recombination operator is very likely to produce offspring with low fitness quality if we want to maintain the feasibility of the solution. With a large number of low fitness quality individuals in the population, the probability of reproducing high fitness quality offspring becomes smaller, and the evolution speed is also decreased. If more offspring with high fitness quality can be produced by the variation operators in each generation, then the efficiency of GA can be greatly improved. However, a possible drawback is that such improvement mechanisms usually result in the production of similar offspring, which decreases the diversity of the population.

In order to obtain a good balance between the fitness quality and the diversity of the offspring, we define the preference information between any two users to guide the operator. The concept is adopted from the CAC algorithm introduced in chapter 3. The mechanism is proven to be useful in aiding the variation operators to produce offspring with high fitness quality while maintaining the diversity of the population.

The definition of the preference information in GGA follows the same manner of the one in CAC algorithm. At the end of each generation of GGA, after new population is evaluated, m individuals with the best fitness quality are selected among the population, and their compositions are memorized to guide the variation operators in the future generations. The method to conserve the compositions is to add a positive value to the preference value between every two users in the same car pool.

Thus, in the last step of the recombination, the operator tries to insert user i into the car pool s which has available car capacity with a roulette wheel selection based on the probability p_{is} calculated in equation (4.1) where w_{is} is the preference between user i and car pool s and N is the set of car pools which have available car capacities. The preference between a user and a car pool is also defined the same as it in the CAC algorithm. If the insertion satisfies the time window constraints, confirm the operation, otherwise, the recombination operator repeat the procedure without the previously ruled out car pool. If all possible car pools are not feasible for the insertion, that is, the time windows are violated after inserting user i, a new car pool will be created for the user.

$$P_{is} = \frac{w_{is}}{\sum_{n \in N} w_{in}} \tag{4.1}$$

4.3.5 Mutation

Four mutation operators are used in GGA, named Divide, Merge, Swap and Reinsert. The Divide, Merge and Swap operators are adopted from the local search procedure of the CAC algorithm. These three operators conserve the same search mechanism as

they are in CAC, except in GGA they randomly select car pools to process instead of selecting based on a specific probability as in CAC, in order to provide larger diversity to the population.

According to the mutation rate, offspring are mutated by applying these operators. Each operator is used with equal probability (25%).

Divide mutation operator

The Divide mutation operator consists in divide several selected car pool into smaller car pools with respects to the total travel cost. The operator selects randomly *p%* of the car pools. Then, for each selected car pool *i*, the operator pools any number of users of pool *i* into a new pool. The operation on car pool *i* ends as soon as the total cost decreases with a feasible result, and the next selected car pool will be processed. The complexity of this operator is O(n).

Merge mutation operator

The Merge mutation operator tries to merge the non-full car pools. For each non-full pool *i*, the operator tries to merge it with any non-full car pool *j* which is able to satisfy the car capacity after merging. The car pool *j* is selected in random order, and the operation is confirmed as soon as a feasible solution with decreased total cost is obtained. If no car pool *j* can be found, the operator will skip car pool *i* and move to the next non-full car pool. The complexity of this operator is O(1).

Swap mutation

The Swap mutation operator chooses randomly *q%* of the car pools. For each selected car pool *i*, the operator selects its nearest car pool *j* according to their gravity centers. Then, it tries to swap every user of pool *i* with every user in pool *j*. The move is confirmed as soon as the total cost decreases with a feasible result. The complexity of this operator is $O(n^2)$.

Reinsert mutation

The Reinsert mutation operator randomly selects *s%* of car pools and randomly removes one user from each car pool. Then the operator reinserts these users back into the car pools with the same insertion procedure of the recombination operator. The probability of selecting a car pool for a user to insert is calculated the same as in equation (4.1). New car pools may be created in order to maintain the feasibility. The complexity of this operator is O(n).

4.3.6 Adaptive Control

In the classic GA, a good diversity level of the population can result in a better best-found solution. However, even with a flexible selection policy, the diversity of

the population declines rapidly. There are many ways to control population diversity, but most of them are achieved by bringing large amount of random elements into the algorithm. In the context of the LTCPP, the randomness plays a very limited role in generating better fitness individuals, since the car capacity and time window constraints are always violated by the random elements.

Therefore, in the GGA, we propose an adaptive control that is based on varying the recombination and mutation rates. The increase of recombination and mutation rates usually promotes the diversity of the population and delays the convergence of the algorithm. The changeable rates can maintain the diversity at a healthy level, a level at which search can progress but does not converge easily. The main idea of the mechanism is to increase the recombination and mutation rates when the population starts to lost its diversity, and decrease the rates when the diversity reaches its upper limit.

The typical way to measure the diversity of a population is to calculate the Hamming distance between any two individuals. However, for our representation, the simple Hamming method is inaccurate, since two same car pools can be located at different places in two individuals. Although a complex method can be designed to compare the individuals, it will greatly increase the computational time, which is not cost-effective.

According to our observation to the population composition when solving the LTCPP instances, a population with large diversity always results in large differences among the fitness values of the individuals in the population. The probability to have a large number of individuals with very different car pool compositions but similar fitness values is relatively very small. Therefore in GGA, we propose an evaluation of the diversity of population by measuring and comparing the mean fitnesses of different portions of the population. This method requires no extra comparison of the individuals; all the needed values are obtained from the survival selection of the population, thus there is no additional computational burden for the algorithm.

After the survival selection, the new population is used to perform the diversity measure, in order to adjust the recombination and mutation rates for the next generation where the new population will be used. The procedure consists in calculating the mean fitness f^{b}_{avg} of the top 20% population which have the best fitness values as well as the mean fitness f^{w}_{avg} of the bottom 20% population which have the worst fitness values. Then, the mechanism modifies the recombination and mutation rates according the comparison of the two mean fitnesses, as shown in equation (4.2). If the two mean fitnesses are close, the diversity of the population is considered to be low. Hence, the recombination and mutation rates will be increased with a small amount. Otherwise, the rates decrease.

$$v = \begin{cases} v' \times (1+\eta) & \text{if} \quad \dfrac{f^{w}_{avg} - f^{b}_{avg}}{f^{w}_{avg}} \le \rho \\ v' \times (1-\eta) & \text{otherwise} \end{cases} \tag{4.2}$$

where $v = \{v_c, v_m\}$ and $v' = \{v'_c, v'_m\}$ are the new and the current recombination and mutation rates, $\eta = \{\eta_c, \eta_m\}$ is the corresponding update value, and ρ is a fixed threshold for the difference between the two mean fitness values, respectively.

In order to avoid generating extreme values, the recombination and mutation rates are limited in control ranges $[v^{min}_c, v^{max}_c]$ and $[v^{min}_m, v^{max}_m]$.

4.3.7 Initialization

The initial population is created with a mix of random and structured individuals without duplication. The expectation is that an initial population of reasonably structured solutions will evolve to high quality individuals in a relatively small number of generations, and drive the search toward optimum. However, a possible drawback is that such a population will lack the diversity. Thus, the random generated initial solutions have also been introduced into the population to prevent the evolution process from converging too quickly.

The method used to generate a population of structured solutions is based on the Sweep Line Algorithm [Souvaine, 2008]. The users are sorted according to increasing order of their horizontal coordinates. To generate each population member, a user is chosen at random to start a new car pool, and then the vertical line sweeps to the right, shown in figure 4.3. The user being swept is allocated to the current car pool if time window and car capacity constraints are satisfied. Otherwise, a new car pool is created for the user. When there are multiple available car pools for a user to be allocated, the user is firstly assigned to the closest car pool. The distance between a user *i* and a car pool is defined to be the average of the distances between user *i* and every existing user *j* in car pool. If the assignment is not feasible, the algorithm tries the next closest one until all available car pools are checked. If the user is still not allocated, a new car pool is created for the user. The process stops temporarily when reach the rightmost user, and continues from the first selected user but sweeps to the left. The algorithm terminates when all users are swept and allocated. If a duplicate solution is obtained, a random car pool will be divided into smaller car pools in order to obtain a new solution. This method has been proved by our experiments to be an effective method of obtaining an initial population of reasonable quality solutions to problems where the users appear in both random and clusters ways.

Note that, when calculating the distance between two users, if we only consider as attributes the coordinates, we are joining in clusters the users that have trips more close in space to each other. But geographic proximity does not guaranty for itself a good match between users because their time schedule may vary, thus the earliest departure time of the users were also introduced into the calculation of distance, shown in equation (4.3).

$$d_{ij} = \alpha \times \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + \beta \times \left| e_i + t_{ij} - e_j \right| \tag{4.3}$$

where x_i, y_i, x_j, y_j are the coordinates of the user i and user j respectively; e_i and e_j are the earliest departure time of user i and user j; α and β are same as the ones for initializing the preference information.

The sweep line algorithm was used to generate x percent of the initial population and the rest of the initial solutions are generated using a random based method. It first generates y one-user car pools formed from randomly selected users. Then, it tries to insert each one of the rest users into these car pools without violating car capacity and time window constraints. The selections of users and car pools are both random. If all existing car pools are infeasible for the user, a new car pool will be created. The same dividing procedure is used to avoid generating duplicate solutions.

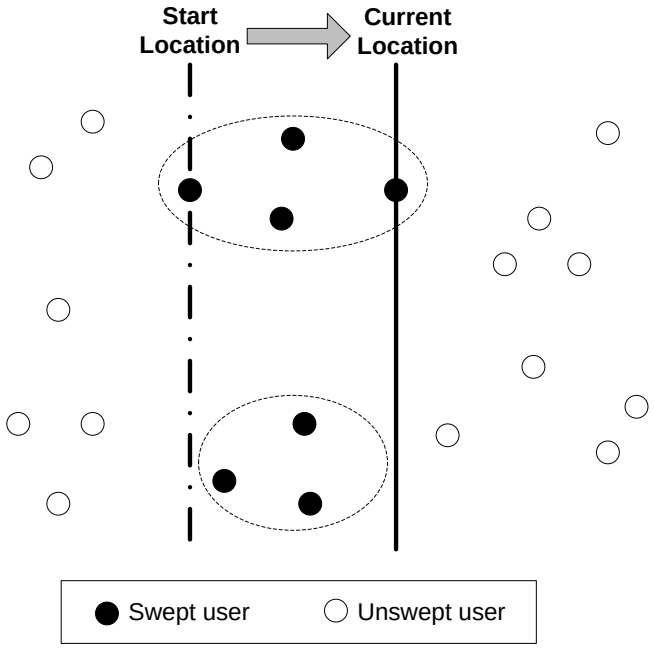


Figure 4.3: Mechanism of the Sweep Line Algorithm.

4.4 Computational Results of GGA

Computational experiments have been conducted to compare the performance of the proposed approach with other approaches.

4.4.1 Configuration

Parameter setting and simulation configuration for the investigated algorithm are specified as follows:

- Population: *120*;
- Initial population: $x = 80$, $y = 10$;
- Initial recombination and mutation rates: $v_{c0} = 0.6$, $v_{m0} = 0.1$;
- Initialization of the preference information: $\alpha = 0.8$, $\beta = 0.2$;
- Update of the preference matrix: *10* best-fit individuals are selected; $\theta = 0.1$;
- Mutation operator setting: $p = 20$, $q = 20$, $s = 20$;
- Adaptive control: $\eta_c = 0.01$, $\eta_m = 0.02$, $\rho = 0.3$;
- Rate control ranges: v_c *[0.6, 0.9]*, v_m *[0.1, 0.4]*.

Given limited computational resources and combinatorial complexity, parameter values were determined empirically over a few intuitively selected combinations, choosing the one that yielded the best average output.

4.4.2 Experimental results

In order to provide an exclusive evaluation of the performance of our algorithm, our experimental results are compared with three other approaches for solving the LTCPP, the ANTS and the simulation-based approach (SB), as well as the CAC algorithm presented in chapter 3. All the three approaches have been proven previously to have the ability to provide good solution quality for some of our benchmarks. Thus, the comparison with these three approaches will be convincing for the evaluation of the GGA.

The approach is implemented in JAVA. The experiments consist in performing 30 simulation runs for each problem instance on Windows operating system with Intel Core i7 740QM 2.9 GHz CPU and 4 GB RAM. During each run, the GGA is set to generate the same amount of solutions as the CAC approach and the ANTS approach did in chapter 3. The SB approach is still set to run until no improvements can be found, since it is CPLEX based approach, the amount of generated solution is uncountable.

Inst	Size	GGA			CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
C101	100	1585.5	1599.3	13	1585.5	1593.4	11	1585.5	1592.9	17	1647.4	1669.2	91
C102	100	1701.9	1712.0	9	1706.8	1728.2	10	1711.4	1748.5	14	1717.5	1724.8	94
C103	100	1513.7	1543.9	12	1508.6	1527.5	11	1512.6	1535.1	18	1532.2	1599.4	85
C201	200	2672.2	2749.4	31	2703.1	2717.7	25	2784.4	2854.2	57	2761.7	2868.6	329
C202	200	2836.7	2876.5	28	2879.2	2892.9	36	2936.1	3004.5	64	3081.7	3114.1	473
C203	200	2716.0	2891.8	42	2769.3	2834.1	29	2845.9	3003.5	58	2975.1	3182.4	394
C401	400	5489.4	5690.6	248	5533.3	5618.6	189	5833.5	6281.4	424	6174.2	6860.3	934
C402	400	4548.3	4786.4	203	4518.2	4760.3	242	4893.5	5153.2	357	5383.7	5524.5	683
C403	400	5909.6	6085.2	295	5930.7	6046.4	271	6125.6	6742.1	511	6675.2	6994.5	1257
Total		28973.3	29935.1	881	29134.7	29719.1	824	30228.5	31915.4	1520	31948.7	33537.8	4340

Table 4.1: Experimental results of set C instances (clustered user distribution).

Table 4.1 compares the experimental results of the C set instances. The GGA is able to provide new best found solutions for 6 instances. However, considering the average solution quality of 30 runs, the GGA outperforms other approaches on only 2 instances.

Table 4.2 shows the percentage the GGA outperforms other approaches on set C instances, in the aspects of average solution quality and computing time. For each instance, the outperforming percentage is calculated as *(other approach's value – GGA's value) / other approach's value*. Each value in table 4.2 is obtained by averaging the outperforming percentages of the three same-size instances. Comparing with ANTS and SB, the GGA can provide better solution quality with less computing time. However, with respect to the average of 30 runs, the GGA's performance is worse than the CAC approach in both solution quality and the computing time.

Set	Size	CAC		ANTS [Maniezzo, 2004]		SB [Correia, 2007]	
		Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
C	100	-0.17%	-5.76%	0.37%	30.86%	2.80%	87.34%
	200	-0.88%	-15.54%	3.88%	43.15%	6.97%	91.33%
	400	-0.82%	-7.99%	8.76%	42.31%	14.47%	73.42%
Avg		-0.62%	-9.76%	4.34%	38.77%	8.08%	84.03%

Table 4.2: Solution quality and computing time comparison on set C instances.

Table 4.3 presents the experimental results on the R set instances. The GGA shows its superiority in solving the random distributed instances. It outperforms other approaches on 8 instances in both best found solution and average solution quality. The results indicate the GGA maintains a good diversity of the population when processing the instances. Table 4.4 reveals the percentage of outperforming in the same manner as table 4.2. Although there is 21.34% computing time gap between the

GGA and CAC according to the calculation method of table 4.4, the computing time of GGA and CAC is still similar considering the summation computing time of 9 instances in table 4.3.

Ints	Size	GGA			CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
R101	100a	2207.1	2235.9	18	2223.1	2283.2	12	2207.1	2281.5	18	2235.1	2265.4	100
R102	100b	1824.5	1867.5	15	1841.4	1874.3	13	1834.6	1864.2	17	1832.8	2091.7	97
R103	100c	2209.1	2286.0	17	2235.9	2313.4	11	2299.2	2438.7	21	2204.7	2418.5	80
R201	200a	4034.8	4188.3	43	4156.1	4231.8	36	4101.5	4253.5	108	4425.0	4567.1	430
R202	200b	3646.8	3751.7	41	3717.2	3824.2	29	3772.2	4071.9	84	3952.4	4283.3	231
R203	200c	3923.2	4158.4	38	4164.7	4304.6	44	4368.5	4541.5	116	4092.4	4257.5	540
R401	400a	7514.9	7799.5	392	7891.4	8033.7	358	8396.1	8580.4	581	8787.8	8993.8	1106
R402	400b	6172.7	6254.0	277	6365.2	6559.7	204	6512.7	6893.3	479	7258.7	7417.5	896
R403	400c	7670.2	7872.9	311	8023.4	8129.5	289	8113.1	8338.9	631	8841.9	8933.5	1037
Total		39203.3	40414.2	1052	40618.4	41554.4	996	42505	44663.9	2055	43630.8	45228.3	4517

Table 4.3: Experimental results of set R instances (random user distribution).

Size	CAC		ANTS [Maniezzo, 2004]		SB [Correia, 2007]	
	Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
100	1.21%	-39.98%	2.69%	10.27%	5.83%	81.76%
200	2.11%	-15.73%	5.94%	59.54%	7.68%	88.40%
400	3.58%	-8.32%	7.99%	47.54%	13.61%	70.90%
Avg	2.30%	-21.34%	5.54%	39.12%	9.04%	80.35%

Table 4.4: Solution quality and computing time comparison on set R instances.

The experimental results for the set W instances (real-world instances) are presented in table 4.5. The GGA provides better results on 6 instances in the average solution quality, and gives new best solutions for 5 instances. The outperforming percentage is shown in table 4.6.

Considering all three sets of instances, the solution quality of GGA is significant better than other approaches when dealing with the instances with random user distribution. However, the CAC is still a better approach for the instances with clustered distributed users. For the small size (100 users) W set instances, the GGA can provide an insignificant improvement with 33.54% more computing time compared with the CAC approach. Therefore, we believe CAC is still a better choice for real-world implementation if the number of users is relatively smaller.

Inst	Size	GGA			CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
W101	100	864.2	879.3	12	864.2	886.3	9	889.4	893.8	18	864.2	885.7	81
W102	100	1000.6	1010.2	14	998.9	1008.5	11	1016.5	1020.3	18	1007.5	1037.9	82
W103	100	1088.2	1126.3	14	1112.4	1134.8	10	1142.4	1168.1	20	1100.5	1187.6	87
W201	200	1528.6	1562.1	35	1523.1	1557.6	46	1586.1	1601.8	97	1717.1	1722.9	341
W202	200	1719.5	1768.5	33	1803.7	1812.9	22	1872.2	1919.2	104	2014.3	2127.7	406
W203	200	1701.2	1743.8	51	1701.2	1784.4	40	1795.6	1907.5	85	1860.5	1965.1	371
W401	400	2608.8	2694.5	311	2789.5	2848.4	356	2843.4	3066.4	481	3168.4	3442.8	955
W402	400	3210.2	3406	402	3283.1	3360.1	324	3541.6	3692.9	441	3633.5	3984.9	792
W501	565	4733.2	4896.6	488	4887.5	5056.2	515	5090.2	5224.9	697	5377.5	5858.3	1621
Total		18454.5	19087.3	1360	18963.6	19449.2	1333	19777.4	20494.9	1961	20743.5	22212.9	4736

Table 4.5: Experimental results of set W instances (real world cases).

Size	CAC		ANTS [Maniezzo, 2004]		SB [Correia, 2007]	
	Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
100	0.46%	-33.54%	2.06%	28.52%	2.85%	84.01%
200	1.48%	-17.86%	6.30%	57.40%	12.49%	89.29%
400	2.40%	-2.06%	8.73%	24.72%	17.56%	62.19%
Avg	1.44%	-17.82%	5.70%	36.88%	10.97%	78.49%

Table 4.6: Solution quality and computing time comparison on set W instances.

Same as the previous chapters, a further evaluation of the performance among the four approaches has been conducted by using a Freidman test. The test consists in the average solution quality on all 27 instances used in our experiments. The detail information is presented in table 4.7, the detail calculation is presented in appendix 2.

The Friedman statistic value T of table 4.7 is calculated to be *46.2*, while the threshold for the F distribution with a significance level 0.01 is *4.04*. Since T is much greater than the threshold, it is proven that there exists at least one approach whose performance is significant different from at least one of the other approach. A paired comparison is then performed to decide which approaches are really different. According to the paired comparison, for significance level 0.01 and 78 degrees of freedom, the critical value for a significant difference between two approaches is *16.25*.

Table 4.8 shows the difference between the performances of every two approaches. According to the values in the table, the GGA outperforms significantly the ANTS and the SB approaches, but the performance difference between GGA and CAC is not significant. However, considering the aspect of best found solution of 30 runs, where the GGA outperforms CAC on 20 instances, the GGA clearly shows its superiority in this aspect.

Instance	GGA			CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]		
	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2
C101	1599.3	3	9	1593.4	2	4	1592.9	1	1	1669.2	4	16
C102	1712	1	1	1728.2	2	4	1748.5	3	9	1724.8	4	16
C103	1543.9	3	9	1527.5	1	1	1535.1	2	4	1599.4	4	16
C201	2749.4	2	4	2717.7	1	1	2854.2	3	9	2868.6	4	16
C202	2876.5	1	1	2892.9	2	4	3004.5	3	9	3114.1	4	16
C203	2891.8	2	4	2834.1	1	1	3003.5	3	9	3182.4	4	16
C401	5690.6	2	4	5618.6	1	1	6281.4	3	9	6860.3	4	16
C402	4786.4	2	4	4760.3	1	1	5153.2	3	9	5524.5	4	16
C403	6085.2	2	4	6046.4	1	1	6742.1	3	9	6994.5	4	16
R101	2235.9	1	1	2283.2	4	16	2281.5	3	9	2265.4	2	4
R102	1867.5	2	4	1874.3	3	9	1864.2	1	1	2091.7	4	16
R103	2286	1	1	2313.4	4	16	2438.7	4	16	2418.5	3	9
R201	4188.3	1	1	4231.8	2	4	4253.5	3	9	4567.1	4	16
R202	3751.7	1	1	3824.2	2	4	4071.9	3	9	4283.3	4	16
R203	4158.4	1	1	4304.6	3	9	4541.5	4	16	4257.5	2	4
R401	7799.5	1	1	8033.7	3	9	8580.4	3	9	8993.8	4	16
R402	6254	1	1	6559.7	2	4	6893.3	3	9	7417.5	4	16
R403	7872.9	1	1	8129.5	2	4	8338.9	3	9	8933.5	4	16
W101	879.3	1	1	886.3	3	9	893.8	4	16	885.7	2	4
W102	1010.2	2	4	1008.5	1	1	1020.3	3	9	1037.9	4	16
W103	1126.3	1	1	1134.8	2	4	1168.1	3	9	1187.6	4	16
W201	1562.1	2	4	1557.6	1	1	1601.8	3	9	1722.9	4	16
W202	1768.5	1	1	1812.9	2	4	1919.2	3	9	2127.7	4	16
W203	1743.8	1	1	1784.4	2	4	1907.5	3	9	1965.1	4	16
W401	2694.5	1	1	2848.4	2	4	3066.4	3	9	3442.8	4	16
W402	3406	2	4	3360.1	1	1	3692.9	3	9	3984.9	4	16
W501	4896.6	1	1	5056.2	2	4	5224.9	3	9	5858.3	4	16
Avg		1.48			1.96			2.93			3.74	
Sum		40	70		53	125		79	243		101	389

Table 4.7: Friedman test results.

$\lvert R_i - R_j \rvert$	CAC	ANTS [Maniezzo, 2004]	SB [Correia, 2007]
GGA	13	39	61
CAC	-	26	48
ANTS [Maniezzo, 2004]	-	-	22

Table 4.8: Paired comparison results.

The accuracy of the GGA is examined by calculating the standard error (column Std) of the solutions obtained in 30 runs of each instance. The solution quality difference (column Diff) between the best found solution quality and the average solution quality of *each* instance in the previous tables is also calculated. Table 4.9 shows the *average* of the abovementioned values of the three same-size instances. The average differences between the best found solution and the average solution of the

GGA on three sets of instances are around 2.8%, which is higher than the CAC approach.

Size	C set instances				R set instances				W set instances			
	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)
100	1600.4	1618.4	12.4	1.1	2080.2	2129.8	22.4	2.3	984.3	1005.3	9.7	2.0
200	2741.6	2839.2	51.9	3.4	3868.3	4032.8	64.9	3.1	1649.8	1691.5	31.4	2.5
400	5315.8	5520.7	93.1	3.7	7119.3	7308.8	125.5	2.5	3517.4	3665.7	78.2	4.0
Avg	3219.3	3326.1	52.5	2.7	4355.9	4490.5	70.93	2.8	2050.5	2120.8	39.8	2.8

Table 4.9: Evaluation of the accuracy of the GGA.

To verify the performance of the adaptive variation rate mechanism, the GGA has also been tested with a fixed variation rate. The average solution qualities of the three same-size instances are shown in the table 4.10. Observe that, the GGA with adaptive variation rate leads in the solution quality consistently in all entries. The results show that the adaptive control plays an important role in improving solution quality and possibly directing search to unknown regions to avoid being trapped in a locality. An example that presents the diversity changing of the search process is shown in figure 4.4.

Size	C set instances		R set instances		W set instances	
	Adaptive	Fixed	Adaptive	Fixed	Adaptive	Fixed
100	1618.4	1652.9	2129.8	2162.8	1005.3	1023.6
200	2839.2	2934.3	4032.8	4218.3	1691.5	1754.3
400	5520.7	5803.4	7308.8	7751.7	3665.7	3866.2
Total	9978.3	10390.6	13471.4	14132.8	6362.5	6644.1

Table 4.10: Evaluation of the adaptive variation rates.

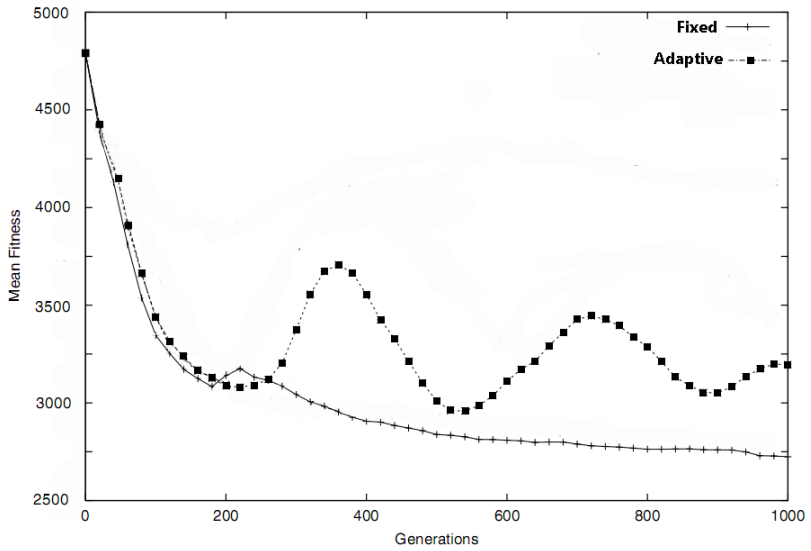


Figure 4.4: Comparison between fixed and adaptive variation rates.

4.4.3 Summary

In section 4.3, we introduced GGA, a new guided genetic algorithm for solving the long-term car pooling problem. We adopt the preference mechanism into the genetic paradigm to aid the variation operators to generate high quality solutions. Furthermore, an adaptive variation rate mechanism is also developed to maintain a better diversity of the population.

The presented approach has been applied successfully for solving the long-term car pooling problem. Experiments have been performed in section 4.4 to confirm the efficiency and the effectiveness of the GGA approach. The approach has been compared with three other metaheuristics for solving the long-term car pooling problem. For most of the instances, the GGA can provide solutions with good quality. The adaptive variation rate mechanism has been proven to be useful. Thus, it has been demonstrated that the GGA algorithm is an effective approach for solving the long-term car pooling problem.

4.5 Multi-agent Self-adaptive Genetic Algorithm for the LTCPP

Based on our previous research, we believe the population based metaheuristics possess some weakness in solving the long-term car pooling problem. First, although the use of metaheuristics allows to significantly reducing the computational complexity of the search process, the latter remains time or memory consuming to obtain the good solution quality for the large size instances. Second, even with the aid of multiple diversification mechanisms, the ability to explore other areas of the search space is significant decreased after the convergence to an optimum. Third, the population-based algorithms require a large number of accurate parameter settings in order to obtain good search ability, thus an inaccurate parameter setting may greatly decrease the performance of an algorithm. At last, the structures of the algorithms are always fixed, thus when a new operator is developed or an old constraint is modified, the whole system structure of the algorithm needs to be redefined, which results in a large work load and a weak system robust. Therefore, we are motivated to develop an improved approach for the LTCPP to overcome these disadvantages.

The first technology we adapt into our new approach is the multi-agent system (MAS). In the multi-agent approach, when a portion of the population searches around a local optimum it may discover, while the other portions of the population continue to search for new local optima, and the process is repeated if any more local

optima are found. The interaction among the agents of MAS is able to maintain the diversity after convergence. Therefore, with the same computational time, the MAS can increase the solution quality obtained by a standard population based metaheuristics. Furthermore, by proceeding with the calculation on multiple workstations, the MAS can improve the computational speed of the algorithm.

The hyper-heuristic is the second technology we added into our improvement approach, in order to develop a more intelligent system to facilitate the complex parameter setting of the standard population based metaheuristics. We define different operators which are able to provide different levels of intensity and diversity to the population, and the hyper-heuristic is used to find the most suitable operator or sequence of operators to apply at an appropriate situation. Thus the design of each individual operator becomes more flexible. Furthermore, with the hyper-heuristic combined with the MAS, any new operator can be easily inserted into the system without modifying the system's main structure, since the hyper-heuristic will select the most appropriate operator to apply.

Thus, we then investigate in this section to merging the Genetic Algorithm with the multi-agent system and the hyper-heuristic. For this purpose, we elaborate a Multi-agent Self-adaptive Genetic Algorithm for solving the long-term car pooling problem.

4.5.1 Multi-agent System and Hyper-heuristic

Multi-agent systems (MAS) is a subfield of Artificial Intelligence (AI) research dedicated to the development of distributed solutions to complex problems regarded as requiring intelligence. Classic AI concepts are modified, so that multi-processor systems and clusters of computers can be applied with new mechanisms, in order to improve the computational speed and the quality of solutions.

The fast development of technology in designing processors, networks, and data storage has made the use of parallel computing more and more popular. Such architectures represent an effective strategy for the design and implementation of parallel metaheuristics. Indeed, sequential architectures are reaching physical limitation. Nowadays, even laptops and workstations are equipped with multicore processors, which represent a given class of parallel architecture. Moreover, the cost performance ratio is constantly decreasing. The proliferation of powerful workstations and fast communication networks have shown the emergence of clusters of processors, networks of workstations, and large-scale network of machines as platforms for high-performance computing. Parallel and distributed computing can be used in the design and implementation of multi-agent metaheuristics for the following reasons:

• Speed up the search: One of the main goals of the multi-agent system is to reduce the search time. This helps designing on-line and interactive optimization methods. This is a very important aspect for the complex optimization problems in real-world

implementation where there is hard requirement on the search time.

• Improve the quality of the obtained solutions: Parallel models for metaheuristics might allow improving the quality of the search. Indeed, exchanging information between cooperative metaheuristics will alter their behavior in terms of searching in the landscape associated with the problem. The main goal of a parallel cooperation between metaheuristics is to improve the quality of solutions. Both better convergence and improvement in the quality of solutions may happen.

• Improve the robustness: A parallel metaheuristic may be more robust in terms of solving in an effective manner different optimization problems and different instances of a given problem by easily inserting, removing, and modifying the operators.

• Solve large size problems: Parallel metaheuristics allow solving large size instances of complex optimization problems, which cannot be solved by a sequential machine.

The typical multi-agent systems, such as Strategies proposed by Cung et al. [1997, 2001], Co-Search [Talbi and Bachelet, 2004] and A-Team [Jedrzejowicz and Wierzbowska, 2006], have been proposed with the above mentioned purposes.

Contrary to the organizations of typical multi-agent systems, the distribution principle of our approach is based on coalition. In the multi-agent field, a coalition is a structure where agents have the same capacities and cooperate by means of direct interactions. Agents contribute to the achievement of the same task. In the above mentioned systems, agents' behaviors correspond to a functional decomposition of the optimization process, but in our system, each agent has its own optimization process, they have the same initial capacities and status. The objective is to exploit cooperation in order to dynamically improve the optimization strategy of the agents. In addition, the coalition structure is intended to support robustness and facilitate the distribution since control is decentralized, communications between agents are asynchronous, and consequently, the removal or addition of any agent will not perturb the global functioning of the system.

The term hyper-heuristic was first created in Denzinger et al. [1997], and it was then used to describe the idea of using heuristics to choose heuristics. The fundamental difference between meta-heuristics and hyper-heuristics is that most implementations of meta-heuristics search within a search space of problem solutions, whereas hyper-heuristics always search within a search space of heuristics. Thus, when using hyper-heuristics, we are attempting to find the right method or sequence of heuristics in a given situation rather than trying to solve a problem directly.

In Burke et al. [2009], hyper-heuristics have been classified according to the source of feedback used by the hyper-heuristic. The classification distinguishes the hyper-heuristics which use online learning, off-line learning and no learning. To be specific, learning refers to the concept of adaptation mechanism in Multi-agent Systems. This learning mechanism modifies the search strategy according to the experiences obtained during the search process. In online learning hyper-heuristics, the

modification takes place while the algorithm solves the problem instance. For off-line learning approaches, the search strategy is defined with a training phase.

4.5.2 General Introduction of the Approach

The multi-agent self-adaptive genetic algorithm (AGA) presented in this paper integrates the multi-agent system, the hyper-heuristic and the genetic structure into one single system. The system is composed of several agents; each agent processes part of the population, and shares its progress with all other agents. The genetic structure is used as a main algorithm structure to provide the evolutionary characteristic for each agent. The hyper-heuristic with both online and off-line learning mechanisms, which is built outside the genetic structure, guides the agents to select the most appropriate genetic operators during each generation. The genetic operators correspond to the recombination and mutation operators of the genetic algorithm, which work on the problem search space and produce solutions. These operators respectively represent different levels of intensity and diversity to the population. That is, the goal of the intensity focused operators is to concentrate the search in the promising areas of the search space. On the contrary, the goal of the diversity focused operators is to move the search to the unexplored areas. A comprehensive study of the concepts of intensification and diversification in metaheuristics can be found in [Blum and Roli, 2003] Note that the operators are easily implemented, removed or replaced, so the AGA is closer to a generally applicable methodology rather than the one solving a single problem instance.

In AGA, an operator pool is defined to store the operators for the agents to select. In the operator pool, the recombination and mutation operators are kept in pairs. Each pair consists of one recombination operator and one mutation operator; the two operators in pair have the same intensity and diversity tendency to the population. The selection of operator pair of each agent is determined by the hyper-heuristic, called decision making heuristic, which is designed to learn adaptively and concurrently guide the behavior of all agents. The decision making heuristic includes two learning phases, see figure 4.5. The first phase, called pre-learning, is an off-line learning process. In this process, the agents are only allowed to use fixed operator pairs and therefore generate an evaluation to each operator pair. This procedure provides initial experiences of the performances of each operator pair in different conditions before the next learning process. This pre-learning phase is performed at the beginning of the approach and takes relatively small percentage of total generations, in order to provide more operating time and spaces for the more intelligent learning phase. The second phase, called reinforcement learning, is an online learning process. In this process, the decision making mechanism is self-adaptive. When an agent gets an improvement with an operator pair under a certain condition, the mechanism modifies the rules of

decision making heuristic according to the level of the improvement, so when the same condition is met again during the genetic process, there will be greater probability for agents to select the same operator pair to modify the search space. This online learning process is designed to favor the operator pair that often improves the current best solution under each condition. In AGA system, all agents share the same operator pool but the decision making heuristic can lead to different strategies of using different operator pairs.

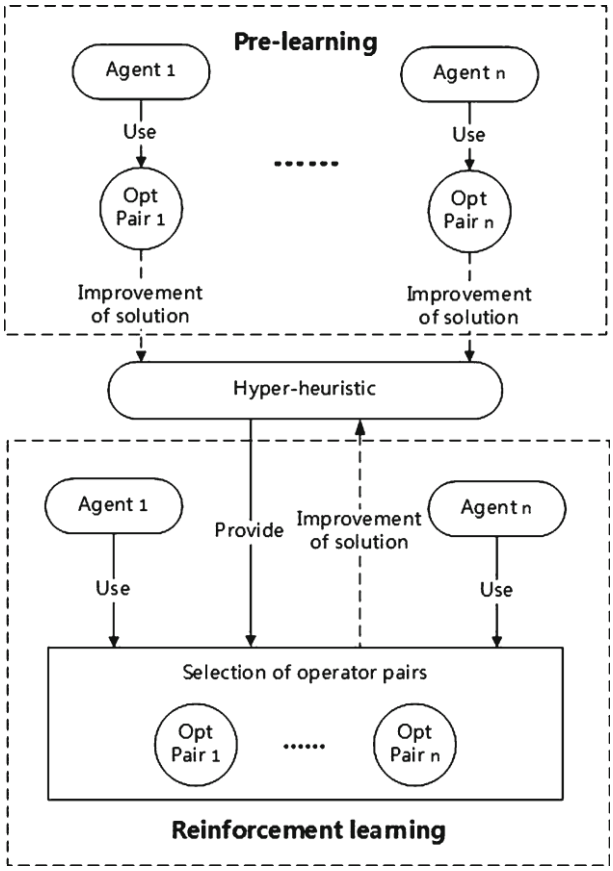


Figure 4.5: Relationship between the learning process and the hyper-heuristic

At the end of each genetic generation of an agent, the migration process is activated. Then, exchanges of some selected individuals between the agents' population are realized, and received individuals are integrated into the local population of each agent. The selection policy of emigrants indicates for each agent in a stochastic way the individuals to be migrated. The stochastic policy does not guarantee that the best individuals will be selected, but its associated computation cost is lower and it is better for the diversity of the population. The number of emigrants is expressed as a fixed number of individuals. The choice of the value of such parameter is crucial. Indeed, if the number of emigrants is low, the migration process will be less efficient as the agents will have the tendency to evolve in an independent way. On the contrary, if the number of emigrants is high, the agents of the system are likely to converge to the same solutions. The integration policy of immigrants is performed at the end of each

generation, before the survival selection. The objective of the mechanism is to delay the global convergence and encourage diversity of the system.

4.5.3 Decision Making Hyper-heuristic

In order to perform the decision making in AGA, a set of rules has been defined in the form of (condition, operator), where the condition indicates the current situation occurred, the operator corresponds to the pair of intensity or diversity focused recombination and mutation operators. Let C be the set of conditions, O the pair of recombination and mutation operators. For a condition c_i, a weight w_{ij} is associated to each pair o_j. The weight w_{ij} corresponds to the likelihood of selection of the pair o_j under the condition c_i. The selection of an operator pair is performed by a roulette wheel selection principle. Thus, the probability $P(c_i, o_j)$ to apply the operator pair o_j in the condition c_i is computed using the following equation (4.4).

$$P(c_i, o_j) = \frac{w_{ij}}{\sum_{n \in N} w_{in}} \tag{4.4}$$

where N is the set of operator pairs.

Decision Making Matrix

	C1	C2	...	Cm
O1	$W_{1,1}$	$W_{2,1}$	...	$W_{m,1}$
O2	$W_{1,2}$	$W_{2,2}$	...	$W_{m,2}$
...	...	...	...	...
On	$W_{1,n}$	$W_{2,n}$	...	$W_{m,n}$

Figure 4.6: Structure of the decision making matrix.

The decision making mechanism can be displayed in the form of a matrix shown in figure 4.6. The columns indicate the different conditions, the rows correspond to the operator pairs, and the blocks show the likelihood of each operator pair being selected in each condition. According to the roulette wheel selection principle, w_{ij} corresponds to the likelihood of selection of operator pair i in condition j. Thus, by increasing or decreasing the weight values, the probability of the operator pairs being selected in certain conditions will be raised or reduced, respectively. The main purpose of the learning mechanism is to improve all agents' choices of operator pair through the modification of the weight values, in order to select the most appropriate operator pair in different conditions. Notice that in AGA, all agents share the same

decision making matrix, this idea is designed to aid the agents to share their experiences easily with each other. This acts as stigmergy behavior which can be found in the swarm intelligence metaheuristics.

The set of conditions is chosen to allow an alternation between different operator pairs. It is defined according to the improvement situation occurred at the end of each generation. Two kinds of solution are used in the condition definition: the local best solution obtained by a certain agent and the global best solution obtained among all agents. The condition set is composed of four different conditions which cover all the situations that may occur in the genetic process:

- C_1: Local or global best solution has been improved in recent m_1 generations; (intensity preferred)
- C_2: Local or global best solution has been improved before more than m_1 but less than m_2 generations ($m_2 > m_1$); (less intensity preferred)
- C_3: No improvement in recent m_2 generations, and no diversification operator pair has been applied in recent m_3 generations ($m_2 > m_3$); (diversity preferred)
- C_4: No improvement in recent m_2 generations, but a diversification operator pair has been applied in recent m_3 generations ($m_2 > m_3$). (intensity preferred)

where m_1, m_2 and m_3 are parameters set by the user according to the total generation number or total run time.

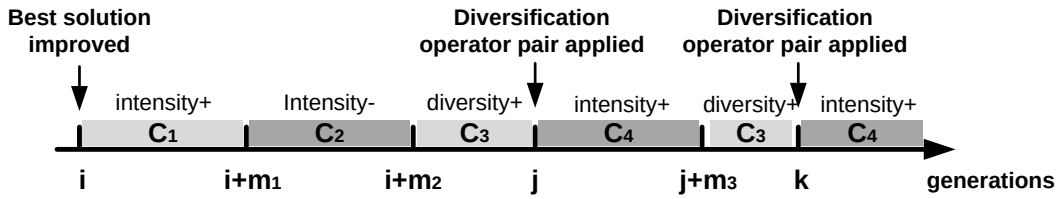


Figure 4.7: The order of the conditions in the search process.

The order of the conditions in the search process is shown in figure 4.7. The condition definition and parameter setting are based on the following considerations: During the first few generations after the global best solution or local best solution has just been improved, it is better now to apply an intensification focused operator pair to search the neighborhood close to the current best solution, which can normally provide high probability in finding further improvement. Thus, an operator pair which provides intensity to the population may be favored from generation i to $i+m_1$. If the operator receives no improvement, we may change to apply another operator pair with less intensity during generation $i+m_1$ to $i+m_2$ to try to improve the current best solution. Afterwards, if there is still no improvement obtained after generation $i+m_2$, the diversification focused operator pair may be used to provide large diversity to the population, by doing which we can receive greater opportunity in finding new improvements. If applying the operator did not bring any improvement, it is possible to

apply another operator pair to explore the neighborhood of the current population for a few generations instead of moving immediately the population to another area of the search space, thus condition C_4 is defined.

It is obvious that a diversification focused operator pair is favored under condition C_3, however the diversification search normally will not immediately improve the current best solution. Thus, in order to reinforce the selection of diversification operator under condition C_3, the following rule is defined.

Definition 4.5.1

When a diversification operator pair is selected and applied under condition C_3, the operator pair will benefit an augmentation of the weight value under this condition, if the current local or global best solution has been improved in the future w generations.

During the genetic process, in the beginning of each generation, the improvement situation of previous generation will be matched with one of the conditions in the condition set. Then the decision making heuristic will proceed to select the appropriate operator pair for this condition, as shown in figure 4.8. In the end of the generation, an evaluation of this selection will be performed. If there is an improvement obtained in this generation, the corresponding weight value in decision making matrix will be increased (except the evaluation of a diversification focused operator pair applied under condition C_3, which will last for *w* generations). In other words, the operator used in the current generation is determined by improvement situation of the previous generation.

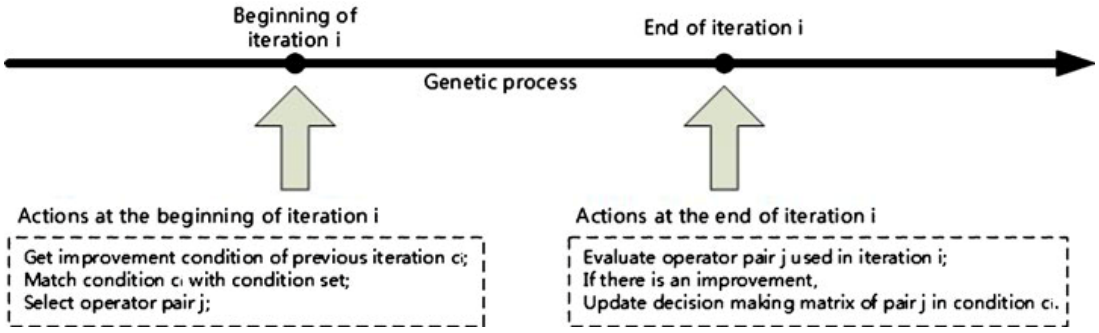


Figure 4.8: Actions in each generation of a genetic process.

4.5.4 Learning Mechanisms

The agents sequentially use both off-line learning and online learning mechanisms to adjust their behaviors, called pre-learning and reinforcement learning as mentioned before. The learning is performed during the genetic process in order to improve the agents' selection strategy of the operator pairs.

Pre-learning mechanism

At the beginning, the decision making matrix is initialized with equal values. The weight values then are updated by the pre-learning phase, where each agent is forced to use only one particular operator pair. If an improvement is obtained under a certain condition, the relative weigh value of this operator pair will be modified synchronously in the matrix.

Thus, in the pre-learning phase of genetic process, agents are disabled from selecting operator pairs, and they are forced to use only the operator pair assigned to them. Vice versa, each operator pair must be used by at least one agent, in order to make sure all operator pairs can be evaluated. Since the amount of agents and operator pairs do not always equal to each other, the pre-learning phase will end until every operator pair is evaluated with equal number of generations. The mechanism to update the decision making matrix is the same as reinforcement learning phase, which will be introduced in detail in the next part.

Reinforcement Learning Mechanism

The reinforcement learning is defined as how an agent ought to take actions in an environment so to maximize some notion of cumulative reward.

In AGA, the problem of selecting the most appropriate operators is viewed as a reinforcement learning problem. During the genetic process, an agent applies different operator pairs in different conditions and learns from the application experiences. To perform the learning, it is necessary to identify the beneficial experiences and determine a reward. This problem is known as the credit assignment problem. An experience is defined as a triplet (condition c_i; operator o_j; improvement v) where the improvement is the fitness difference obtained by the application of the operator pair. When an implementation of an operator pair made an improvement of the global or local best solution, the reinforcement learning procedure consists in an augmentation by a factor σ of the weight value w_{ij}. That is, when a new best local or global solution has been obtained, the operator pair of this generation is reinforced. This mechanism is intended to favor the operator pair that often finds new best solutions in a certain condition. If the best found solution is not changed in this generation, then the weight matrix will not be modified.

In order to refine the reinforcement learning process, two cases are distinguished, (1) When the agent improves its local best solution, and (2) when the agent improves the global best solution in addition to the local best solution. The learning factors σ_1 and σ_2 are respectively used for these two cases. Before adding the reinforcement value to the weight value w_{ij} of applying operator pair o_j under condition c_i, all existing weight values w_{ij} related to the corresponding condition c_i in the decision making matrix will decrease with a rate μ, in order to enlarge the influence of the new reward obtained in the current generation. The reinforcement is performed using equations (4.5) and (4.6).

$$w'_{ij} = \mu \times w''_{ij} \tag{4.5}$$

$$w_{ij} = w'_{ij} + \sigma \tag{4.6}$$

where w'_{ij} is the weight value before adding the reinforcement, w''_{ij} is the weight value before the evaporation, and σ: $\{\sigma_1, \sigma_2\}$ is the learning factor, respectively.

Figure 4.9 presents a simple case where reinforcement learning procedure is applied. Suppose the current condition is c_1 with the setting: Local or global best solution has been improved in recent *5* generations. In this condition, operator o_1 is applied for the current generation and obtained an improvement which decreases the local best solution from *3200* to *3100*. Then, reinforcement is applied based on the experience *(c_1, o_1, 100)*. Thus, the weights w_{11} is augmented to favor the selection of the operator pair in this condition.

In figure 4.9, the original weight matrix is shown on the left, and the matrix on the right obtained a reinforcement with $\sigma = 1$. The reinforcement procedure clearly affects the next selections of the operator pairs. If condition c_1 is met again after the reinforcement, then operator o_1 has *51% (2.8/5.5)* probability to be selected against *40% (2/5)* before the reinforcement.

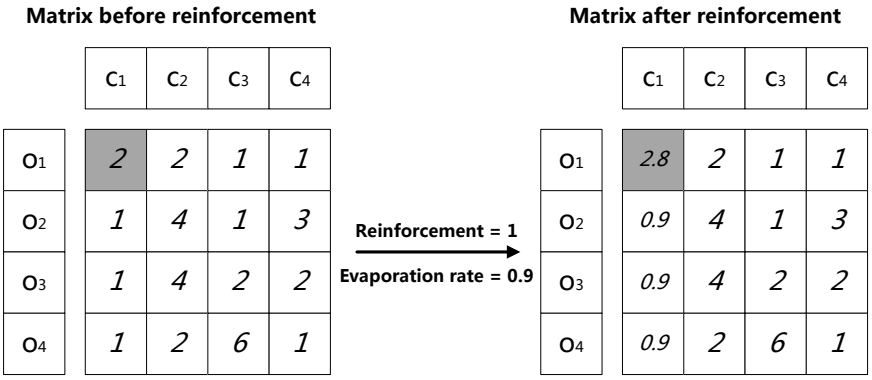


Figure 4.9: Reinforcement learning procedure.

The combination of pre-learning and reinforcements learning allows to introduce adaption into the population based search, and then to enhance individual and global behavior. An agent exploits its past experiences in order to improve its ability to find new best solutions; it also shares its experiences in order to collectively ensure better actions of the system in the future.

4.5.5 Main Structure of the AGA

The behavior of AGA agents is based on four components: genetic structure, operator pairs, decision making heuristic and learning mechanisms. The genetic structure provides the evolutionary procedure. The operators are designed to proceed with intensi-

fication focused or diversification focused searches. The decision making heuristic determines the selection of operator pairs while maintaining a genetic generation. For each application of an operator pair, the agent's population is examined and if it is beneficial, the operator pair is rewarded. Based on the experiences accumulated during each generation, learning mechanisms modify the rules of the decision making process.

The behavior of AGA agents is described in algorithm 4.3. In the pre-learning phase, in the beginning of each generation, the improvement situation of previous generation will be matched with one of the conditions in the condition set. Then the assigned operator pair will be applied. If the agent improves the best solution, the learning mechanism will be activated, and then the decision making matrix will be updated. In the reinforcement learning phase, the procedure starts by verifying the improvement condition of the previous generation, then selects and applies an operator according to the condition. Then, same as pre-learning phase, if the agent has made an improvement, the reinforce mechanism will be executed. Note that, the global best solutions and the decision making matrix are always shared immediately during the process, there is no synchronization point in the system.

The migrations are performed inside a ring topology. Each agent has a source agent for receiving individuals and a destination agent for sending the emigrant individuals. Another element with important consequences over the algorithm is the asynchronous migration parameterization. Frequent migrations may result in a premature convergence while rare migrations fall in the opposite case. In our case, *z%* of the population migrate in asynchronous manner, that is, migrations occur at different times. An agent migrates its individuals at the end of its generation, but the emigrants are accepted when the corresponding agent reach its own end of generation. The choice of asynchronous communication is related to the fact that the speedup performance of the system is expected to be higher than synchronous system. Indeed, in the synchronous model, the evolution process is often hanging on powerful machines waiting the less powerful ones to complete their computation. A stochastic binary tournament selection strategy is being applied for selecting the emigrant individuals where the better individuals are selected to migrate to the population of the target agent.

Algorithm 4.3*: Multi-agent Self-adaptive Genetic Algorithm*

Agent n

Generate initial population;
Initialize the guidance information;
Assign operator pairs to agents;

/ Pre-learning phase */*

While *stopping criterion of pre-learning is not met* ***do***
Get the improvement condition of previous generation c_i*;*
While *not exceed the recombination rate* ***do***
Parent selection;
Apply assigned operator pair o_j*;*
End while
Get the new best found solution of current generation;
If *the global or local best solution is improved;*
Update the global or local best solution;
Update decision making matrix of (c_i , o_j)*;*
If *a diversification operator pair* o_k *was applied under condition* c_3 *in recent w generations, update decision making matrix of* (c_3 , o_k)*;*
Migrate z% of the population to agent n+1;
If *there is migrators from agent n-1, then receive;*
Survival selection;
End While

/ Reinforcement learning phase */*
While *stopping criterion of reinforcement learning is not met* ***do***
Match condition c_i *with condition set;*
Choose operator pair o_j *according to decision making matrix* (c_i , o_j)*;*
While *not exceed the recombination rate* ***do***
Parent selection;
Apply operator pair o_j*;*
End while
Get the new best found solution of current generation;
If *the global or local best solution is improved;*
Update the global or local best solution;
Update decision making matrix of (c_i , o_j)*;*
If *a diversification operator pair was applied* o_k *under condition* c_3 *in recent w generations, update decision making matrix of* (c_3 , o_k)*;*
Migrate z% of the population to agent n+1;
If *there is migrators from agent n-1, then receive;*
Survival selection;
End While

4.5.6 Parallel implementation

In the parallel implementation of the algorithm, all agents share the same decision making matrix. Each agent can modify the rules of the decision process according to the agent's experiences, so it can share the information with the other agents.

There are two information transfers among the agents. On one hand, if the global best solution has been improved by an agent, it will broadcast the information to other agents. On the other hand, at the end of each iteration, part of the population of one agent will migrate to another agent in order to enlarge the diversity of the populations of the agents.

4.5.7 Specialization for the LTCPP

The specialization of AGA for the Long-term Car Pooling Problem requires the definition of the diversification and intensification focused operator pairs particularly designed for the LTCPP. Based on Guided Genetic Algorithm presented in section 4.3, four pairs of operators are defined in the operator pool, including four recombination operators and four mutation operators. They are designed to provide different levels of intensity and diversity tendency to the population. To be more specific, two pairs are defined to focus on the intensity tendency; however, they are given different intensity levels. The other two pairs of operators are intended to provide the diversity to the population, and they also have different diversity levels. It is important to note that the recombination and mutation rates are associated to each individual operator pair. That is, if an operator pair is selected by an agent, then the agent must generate the genetic operations with the specific recombination and mutation rates of this pair. So when defining the recombination and mutation operator pairs, the correlative recombination and mutation rates have to be defined at the same time. The operator pairs and other details of specialization are introduced as follows.

4.5.7.1 Solution representation and initial solution

The direct encoding is still used in this approach. The chromosome is encoded in the same manner as the Guided Genetic Algorithm. The detail is introduced in section 4.3.2.

The initial population contains both random and structured solutions. The expectation is that an initial population of reasonably structured solutions will evolve to high-quality solutions in a relatively small number of generations. However, a possible drawback is that such a population will lack the diversity needed to obtain near-optimal solutions. Thus, the randomly generated initial solutions have also been considered. Both types of initial solutions possess half of the population. The method to generate the initial solution is also same as the Guided Genetic Algorithm, which is presented in section 4.3.6.

4.5.7.2 Recombination Operators

Inspired by the adaptive recombination and mutation rates of the Guided Genetic Algorithm, in the operators' pool, four pairs of different operators are defined with different variation rates. Each pair consists in a recombination operator and a mutation operator. When the agent selects the operators during the genetic process, it must select an operator pair, not single recombination or mutation operator. The operator pairs are fixed after the definition, that is, the recombination and mutation operators in each pair will not change during the whole AGA process. The recombination and mutation rates associated with each pair are also fixed. All four recombination operators defined in AGA use the same 2-point recombination mechanism as the Guided Genet-

ic Algorithm, however, the distance $d_{2\text{-point}}$ between the two points in the chromosome are fixed, and each point must be a start or an end of a car pool. Then, one offspring is generated by combining the genes from the first parent which are between the two chosen points in the chromosome, along with the genes from the second parent which are to the left of the first point and to the right of the second point of the chromosome. The duplicate users caused by the recombination will be removed from the second part, and the users which do not exist in any part of genes will be inserted in to the chromosome based on the preference mechanism same as the GGA approach. A second offspring is produced by swapping round the parents then using the same procedure.

The differences among the four recombination operators are the values of $d_{2\text{-point}}$. A $d_{2\text{-point}}$ value close to half of the total length of the chromosome, results in a large number of duplicate users from the two parents need to be removed and reinserted, which provides a high level of diversity between the parents and the offspring. In contrary, a relatively small $d_{2\text{-point}}$ value brings very few duplicate users to remove and reinsert, thus it provides an intensity tendency to the population. Thus, by giving the four recombination operators different $d_{2\text{-point}}$ values, the levels of intensity and diversity provided to the population by the operators are distinguished. Therefore, we define two recombination operators with relatively small $d_{2\text{-point}}$ values, and combined them with the intensification focused mutation operators, in order to form two intensification focused operator pairs. Also, two recombination operators with relatively large $d_{2\text{-point}}$ values are combined with two diversification focused mutation operators to generate two diversification focused operator pairs.

4.5.7.3 Mutation Operators

Four different mutation operators are also defined in the operator pool. The operators are extended from the mutation operators in GGA. We generate two new intensification focused mutation operators, named with M1 mutation and M2 mutation, as well as two new diversification focused operators, named with M3 mutation and M4 mutation. Each mutation operator is associated with one recombination operator to form an operator pair. They are matched according to the intensity and diversity they bring to the population.

M1 and M2 mutation operators (intensification focused)

The M1 and M2 mutation operator choose randomly n_1 and n_2 car pools. For each selected car pool i, the operators select its nearest car pool j according to their gravity centers. Then, M1 mutation operator swaps every user in car pool i and every user in car pool j, while the M2 mutation operator tries to move every user of pool i to any car pool j which has available car capacity. The move is confirmed as soon as the total cost decreases with a feasible result. The complexity of these two operators is $O(n^2)$.

M3 and M4 mutation operators (diversification focused)

The M3 and M4 mutation operators randomly select n_3 and n_4 car pools and remove

randomly one user from each car pool. Then the M3 mutation operator reinserts these users into the non-full car pools with the same insertion procedure of the recombination operator, while the M4 mutation operator reinserts the users randomly into the non-full car pools. New car pools may be created in order to maintain the feasibility. In order to focus on the diversity of the population, the M3 and M4 mutation operators are regardless of the total cost. Since the insertion operation of M3 mutation operator is guided by the preference mechanism and the one of M4 mutation operator is random, the latter can provide more diversity to the population than the former. The complexity of these two operators is O(n).

4.6 Computational Results of AGA

Computational experiments have been conducted to compare the performance of the proposed AGA approach with other approaches.

4.6.1 Configuration

Parameter setting and simulation configuration for the investigated algorithm are specified as follows. These parameters were determined experimentally over a set of combinations, choosing the one that yielded the best average output.

- Number of agents: 4;
- Population: *30*; *z = 10%*;
- Condition definition: $m_1 = 5$, $m_2 = 10$, $m_3 = 5$, $w = 5$;
- Decision making matrix update: $\sigma_1 = 1$, $\sigma_2 = 2$, $\mu = 0.95$;
- Operator pair 1 (intensification focused):
  Recombination operator with $d_{2\text{-}point}$ *= 10% of all car pools*;
  M1 mutation with n_1 *= 20% of all car pools*;
  Recombination rate $cr_1 = 0.6$, mutation rate $mr_1 = 0.4$.
- Operator pair 2 (intensification focused):
  Recombination operator with $d_{2\text{-}point}$ *= 20% of all car pools*;
  M2 mutation with n_2 *= 20 % of all car pools*;
  Recombination rate $cr_2 = 0.6$, mutation rate $mr_2 = 0.2$.
- Operator pair 3 (diversification focused):
  Recombination operator with $d_{2\text{-}point}$ *= 30% of all car pools*;
  M3 mutation with n_3 *= 30% of all car pools*;
  Recombination rate $cr_3 = 0.7$, mutation rate $mr_3 = 0.2$.
- Operator pair 4 (diversification focused):
  Recombination operator with $d_{2\text{-}point}$ *= 40% of all car pools*;
  M4 mutation with n_4 *= 30% of all car pools*;
  Recombination rate $cr_4 = 0.8$, mutation rate $mr_4 = 0.4$.

4.6.2 Experimental results

The AGA is compared with three other approaches for solving the LTCPP, the ANTS and the CAC introduced in chapter 3, as well as the GGA algorithm presented previously in this chapter. All three approaches have been proven to have the ability to provide good solution quality for some of our benchmarks. Thus, the comparison with these three approaches will be convincing for the evaluation of the AGA.

The experiments of AGA consist in performing 30 simulation runs for each problem instance using multi-thread computing technology [Hyde, 1999] on Windows operating system with Intel Core i7 740QM 2.9 GHz CPU and 4 GB RAM. Each agent is simulated with one thread. The AGA approach is still set to generate the same amount of solutions as the other approaches did in the previous chapters.

Table 4.11 compares the experimental results of the C set instances. Considering the average solution quality of 30 runs, the AGA outperforms other approaches on all instances. Furthermore, the GGA is able to provide new best found solutions for 4 instances.

Inst	Size	AGA			GGA			CAC			ANTS [Maniezzo, 2004]		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
C101	100	1585.5	1585.5	5	1585.5	1599.3	13	1585.5	1593.4	11	1585.5	1592.9	17
C102	100	1701.9	1704.1	4	1701.9	1712	9	1706.8	1728.2	10	1711.4	1748.5	14
C103	100	1508.6	1511.6	5	1513.7	1543.9	12	1508.6	1527.5	11	1512.6	1535.1	18
C201	200	2626.8	2671.5	11	2672.2	2749.4	31	2703.1	2717.7	25	2784.4	2854.2	57
C202	200	2806.7	2811.9	10	2836.7	2876.5	28	2879.2	2892.9	36	2936.1	3004.5	64
C203	200	2716	2724.6	13	2716	2891.8	42	2769.3	2834.1	29	2845.9	3003.5	58
C401	400	5425.9	5448.9	84	5489.4	5690.6	248	5533.3	5618.6	189	5833.5	6281.4	424
C402	400	4518.2	4538.0	71	4548.3	4786.4	203	4518.2	4760.3	242	4893.5	5153.2	357
C403	400	5725.9	5796.2	102	5909.6	6085.2	295	5930.7	6046.4	271	6125.6	6742.1	511
Total		28615.6	28792.3	305	28973.3	29935.1	881	29134.7	29719.1	824	30228.5	31915.4	1520

Table 4.11: Experimental results of set C instances (clustered user distribution).

Table 4.12 shows the percentage the AGA outperforms other approaches on set C instances, in the aspects of average solution quality and computing time. For each instance, the outperforming percentage is calculated as *(other approach's value – AGA's value) / other approach's value*. Each value in table 4.12 is obtained by averaging the outperforming percentages of the three same-size instances. Comparing with other approaches, the AGA can provide better solution quality in much less computing time.

Set	Size	GGA		CAC		ANTS [Maniezzo, 2004]	
		Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
C	100	1.14%	58.48%	0.98%	56.36%	1.51%	71.41%
	200	3.62%	65.95%	2.79%	61.13%	7.37%	80.89%
	400	4.73%	65.53%	3.94%	62.86%	13.07%	80.11%
Avg		3.16%	63.32%	2.57%	60.12%	7.32%	77.47%

Table 4.12: Solution quality and computing time comparison on set C instances.

Table 4.13 presents the experimental results on the R set instances. The AGA is still superior in solving the random distributed instances. It outperforms other approaches on all instances in average solution quality, and provides 4 new best found solutions. The results indicate the AGA maintains a good diversity of the population when processing the random distributed instances. Table 4.14 reveals the percentage of outperforming. Note that AGA is better than other approaches in the aspects of both solution quality and computing time.

Ints	Size	AGA			GGA			CAC			ANTS [Maniezzo, 2004]		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
R101	100a	2207.1	2214.7	6	2207.1	2235.9	18	2223.1	2283.2	12	2207.1	2281.5	18
R102	100b	1824.5	1835.7	6	1824.5	1867.5	15	1841.4	1874.3	13	1834.6	1864.2	17
R103	100c	2200.3	2226.9	7	2209.1	2286	17	2235.9	2313.4	11	2299.2	2438.7	21
R201	200a	3966.2	4117.1	12	4034.8	4188.3	43	4156.1	4231.8	36	4101.5	4253.5	108
R202	200b	3646.8	3666.7	15	3646.8	3751.7	41	3717.2	3824.2	29	3772.2	4071.9	84
R203	200c	3923.2	3982.1	14	3923.2	4158.4	38	4164.7	4304.6	44	4368.5	4541.5	116
R401	400a	7354.2	7405.1	143	7514.9	7799.5	392	7891.4	8033.7	358	8396.1	8580.4	581
R402	400b	6172.7	6222.5	102	6172.7	6254	277	6365.2	6559.7	204	6512.7	6893.3	479
R403	400c	7602.0	7695.0	125	7670.2	7872.9	311	8023.4	8129.5	289	8113.1	8338.9	631
Total		38897.0	39365.8	430	39203.3	40414.2	1052	40618.4	41554.4	996	42505	44663.9	2055

Table 4.13: Experimental results of set R instances (random user distribution).

Size	GGA		CAC		ANTS [Maniezzo, 2004]	
	Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
100	1.75%	61.83%	2.93%	46.74%	4.38%	66.01%
200	2.74%	66.22%	4.77%	61.04%	8.49%	86.32%
400	2.61%	62.17%	6.10%	55.60%	10.38%	78.09%
Avg	2.36%	63.41%	4.60%	54.46%	7.75%	76.81%

Table 4.14: Solution quality and computing time comparison on set R instances.

The experimental results for the set W instances (real-world instances) are presented in table 4.15. The AGA provides better results on 8 instances in the average solution quality, and gives new best solutions for 4 instances. The outperforming percentage is shown in table 4.16.

Considering all three sets of instances, the solution quality of AGA is significant better than other approaches in both the best found solution quality and the average solution quality. Moreover, considering the total computing time of all the instances, the AGA is fastest approach among the candidates.

Inst	Size	AGA			GGA			CAC			ANTS [Maniezzo, 2004]		
		Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
W101	100	864.2	868.2	5	864.2	879.3	12	864.2	886.3	9	889.4	893.8	18
W102	100	998.9	1010.5	5	1000.6	1010.2	14	998.9	1008.5	11	1016.5	1020.3	18
W103	100	1088.2	1096	5	1088.2	1126.3	14	1112.4	1134.8	10	1142.4	1168.1	20
W201	200	1523.1	1532	14	1528.6	1562.1	35	1523.1	1557.6	46	1586.1	1601.8	97
W202	200	1719.5	1739.6	13	1719.5	1768.5	33	1803.7	1812.9	22	1872.2	1919.2	104
W203	200	1646.1	1696.6	14	1701.2	1743.8	51	1701.2	1784.4	40	1795.6	1907.5	85
W401	400	2577.2	2608.6	107	2608.8	2694.5	311	2789.5	2848.4	356	2843.4	3066.4	481
W402	400	3188.9	3273	128	3210.2	3406	402	3283.1	3360.1	324	3541.6	3692.9	441
W501	565	4685.1	4759.2	176	4733.2	4896.6	488	4887.5	5056.2	515	5090.2	5224.9	697
Total		18291.2	18583.7	467	18454.5	19087.3	1360	18963.6	19449.2	1333	19777.4	20494.9	1961

Table 4.15: Experimental results of set W instances (real world cases).

Size	GGA		CAC		ANTS [Maniezzo, 2004]	
	Cost Gap	Time Gap	Cost Gap	Time Gap	Cost Gap	Time Gap
100	1.31%	62.30%	1.75%	49.66%	3.33%	73.15%
200	2.09%	64.39%	3.54%	58.49%	8.26%	85.53%
400	3.30%	65.90%	5.63%	65.42%	11.74%	74.49%
Avg	2.23%	64.19%	3.64%	57.86%	7.78%	77.72%

Table 4.16: Solution quality and computing time comparison on set W instances.

As in the previous chapters, a further evaluation of the performance among the four approaches has been conducted by using a Freidman test. The test consists in the average solution quality on all 27 instances used in our experiments. The detail information is presented in table 4.17.

The Friedman statistic value T of table 4.17 is calculated to be *90.73*, while the threshold for the F distribution with a significance level 0.01 is *4.04*. Since T is much greater than the threshold, it is proven that there exists at least one approach whose performance is significant different from at least one of the other approach.

A paired comparison is then performed to decide which approaches are really different. According to Friedman test, for significance level 0.01 and 78 degrees of freedom, the critical value for a significant difference between two approaches is *12.49*. Table 4.18 shows the difference between the performances of every two approaches. According to the values in the table, the AGA outperforms significantly the other approaches.

Instance	AGA			GGA			CAC			ANTS [Maniezzo, 2004]		
	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2
C101	1585.5	1	1	1599.3	4	16	1593.4	3	9	1592.9	2	4
C102	1704.1	1	1	1712	2	4	1728.2	3	9	1748.5	4	16
C103	1511.6	1	1	1543.9	4	16	1527.5	2	4	1535.1	3	9
C201	2671.5	1	1	2749.4	3	9	2717.7	2	4	2854.2	4	16
C202	2811.9	1	1	2876.5	2	4	2892.9	3	9	3004.5	4	16
C203	2724.6	1	1	2891.8	3	9	2834.1	2	4	3003.5	4	16
C401	5448.9	1	1	5690.6	3	9	5618.6	2	4	6281.4	4	16
C402	4538	1	1	4786.4	3	9	4760.3	2	4	5153.2	4	16
C403	5796.2	1	1	6085.2	3	9	6046.4	2	4	6742.1	4	16
R101	2214.7	1	1	2235.9	2	4	2283.2	4	16	2281.5	3	9
R102	1835.7	1	1	1867.5	3	9	1874.3	4	16	1864.2	2	4
R103	2226.9	1	1	2286	2	4	2313.4	3	9	2438.7	4	16
R201	4117.1	1	1	4188.3	2	4	4231.8	3	9	4253.5	4	16
R202	3666.7	1	1	3751.7	2	4	3824.2	3	9	4071.9	4	16
R203	3982.1	1	1	4158.4	2	4	4304.6	3	9	4541.5	4	16
R401	7405.1	1	1	7799.5	2	4	8033.7	3	9	8580.4	4	16
R402	6222.5	1	1	6254	2	4	6559.7	3	9	6893.3	4	16
R403	7695	1	1	7872.9	2	4	8129.5	3	9	8338.9	4	16
W101	868.2	1	1	879.3	2	4	886.3	3	9	893.8	4	16
W102	1010.5	3	9	1010.2	2	4	1008.5	1	1	1020.3	4	16
W103	1096	1	1	1126.3	2	4	1134.8	3	9	1168.1	4	16
W201	1532	1	1	1562.1	3	9	1557.6	2	4	1601.8	4	16
W202	1739.6	1	1	1768.5	2	4	1812.9	3	9	1919.2	4	16
W203	1696.6	1	1	1743.8	2	4	1784.4	3	9	1907.5	4	16
W401	2608.6	1	1	2694.5	2	4	2848.4	3	9	3066.4	4	16
W402	3273	1	1	3406	3	9	3360.1	2	4	3692.9	4	16
W501	4759.2	1	1	4896.6	2	4	5056.2	3	9	5224.9	4	16
Avg	1.074			2.444			2.704			3.778		
Sum		29	35		66	172		73	209		102	394

Table 4.17: Friedman test results.

$\lvert R_i - R_j \rvert$	GGA	CAC	ANTS [Maniezzo, 2004]
AGA	37	44	73
GGA	-	7	36
CAC	-	-	29

Table 4.18: Paired comparison results.

The accuracy of the AGA is examined by calculating the standard error (column Std) of the solutions obtained in 30 runs of each instance. The solution quality difference (column Diff) between the best found solution quality and the average solution quality of *each* instance in the previous tables is also calculated. Table 4.19 shows the *average* of the abovementioned values of the three same-size instances. The average differences between the best found solution and the average solution of the

AGA on three sets of instances are 0.5%, 1.2% and 1.4%, which are the most accurate values we obtained during our research.

Size	C set instances				R set instances				W set instances			
	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)
100	1598.7	1600.4	1.2	0.1	2077.3	2092.4	7.2	0.7	983.8	991.6	3.7	0.8
200	2716.5	2736.0	15.3	0.7	3845.4	3922.0	31.1	2.0	1629.6	1656.1	8.9	1.6
400	5223.3	5261.0	22.1	0.7	7043.0	7107.5	33.8	0.9	3483.7	3546.9	29.1	1.8
Avg	3179.5	3199.1	12.9	0.5	4321.9	4374.0	24	1.2	2032.4	2064.9	13.9	1.4

Table 4.19: Evaluation of the accuracy of the AGA.

To evaluate the performance of hyper-heuristic, two other multi-agent based genetic algorithms MGAR and MGAF are applied. The MGAR corresponds to a multi-agent system with the same number of agents as AGA, the difference is the MGAR has no learning mechanism and the operator pairs are randomly selected by each agent. The MGAF is similar to MGAR except each agent is fixed with one operator pair. The average solution qualities of the three same-size instances are shown in the table 4.20. Observe that, the AGA leads in the solution quality consistently in all instances. The results reveal that the hyper-heursitic plays an important role in selecting the approriate operator pair and improving the solution quality.

Size	C set instances			R set instances			W set instances		
	MGAR	MGAF	AGA	MGAR	MGAF	AGA	MGAR	MGAF	AGA
100	1608.2	1607.2	1600.4	2126.7	2118.1	2092.4	1003.8	998.2	991.5
200	2849.6	2834.9	2736	4012.2	4099.1	3921.9	1695.4	1683.7	1656.7
400	5487.0	5431.3	5261	7330.9	7232.6	7107.5	3681.9	3766.7	3546.3
Total	9944.8	9873.4	9597.4	13469.8	13449.8	13121.8	6381.1	6448.6	6194.5

Table 4.20: Evaluation of the hyper-heuristic.

	C_1 (%)	C_2 (%)	C_3 (%)	C_4 (%)
O_1	62	4	3	31
O_2	18	30	11	41
O_3	12	36	28	24
O_4	11	16	67	6

Table 4.21: Frequency of usage of each operator pair.

Table 4.21 reveals an example of the frequency of usage of each operator pair under different conditions when solving a set C instance. The results show the intelligence of our approach. When a condition occurred, a specific operator pair is favored. The search tendency of each operator pair is proven to be useful; each specific operator pair has contributed to the improvement of solutions in their related condition. For instance, the operator pair one is mostly used under condition c_1, while the operator

pair four is mostly applied under condition c_3. The operator pairs two and three are both favored under condition c_2, since these two operator pairs are defined to provide average intensity and diversity to the population.

4.6.3 Summary

In section 4.5 and section 4.6, we have introduced AGA, a new multi-agent based self-adaptive genetic algorithm to solve the LTCPP. AGA is composed of several agents which concurrently explore the search space but cooperate to coordinate the search and improve their behaviors. Each agent manages a relatively independent genetic algorithm. In each genetic algorithm, the agent applies recombination and mutation operators which are selected by an adaptive decision making mechanism. This decision making mechanism is based on the heuristic rules which are adapted during the optimization process by learning mechanisms.

AGA exploits the combination of multi-agent system, hyper-heuristic and genetic algorithm. The multi-agent system encourages modularity and reusability. The cooperation structure is intended to support robustness and facilitate the distribution, since the control is decentralized and the agents' interactions are asynchronous. The hyper-heuristic with both online and off-line learning mechanisms provides good adaption characteristic and effectively guides the selection of operators in the genetic algorithm.

Some additional criteria such as flexibility and modularity have to be considered since AGA addresses these issues. Flexibility can be defined as the capacity of adapting an algorithm to effectively deal with additional constraints, an algorithm which is highly problem dependent cannot be considered as flexible. Considering this criterion, AGA has several advantages. First of all, new intensity and diversity focused operators can be easily introduced without modifying the architecture of the agents. These operators are automatically managed by the decision making process and learning mechanisms. Then, by using the multi-agent model, others decision or learning procedures can be considered. Finally, the decentralization in AGA and the asynchronous nature of agents' interactions make AGA a good candidate for a parallel execution.

The presented approach has been applied successfully for solving long-term car pooling problem. Experiments have been performed to confirm the efficiency of the system. It is shown that AGA can provide solutions with good quality efficiently when dealing with both small and large scale instances. Also the process time of AGA is less comparing with the other metaheuristics. Thus, it has been demonstrated that AGA is an effective approach for solving long-term car pooling problem and it is competitive with some of the most powerful heuristics.

4.7 Conclusion

In this chapter we introduced a guided genetic algorithm and a multi-agent self-adaptive genetic algorithm for the long-term car pooling problem.

The GGA is designed with an adaptive control of the variation rates. The algorithm adapts the recombination rate and mutation rate dynamically, in order to maintain stable population diversity at a desirable level.

The AGA is designed using a multiple agent framework with a genetic paradigm. It adopts the metaphor of the coalition. Several agents concurrently explore the search space, and they cooperate to coordinate the search and improve their behaviors. To perform the search, an agent uses several genetic operator pairs which are scheduled by an adaptive decision process. The decision process is based on heuristic rules and follows the hyper-heuristic approach which is problem independent. In addition, the decision rules of the agents are adapted during the optimization process by a learning mechanism. The coalition structure is intended to support robustness and facilitate the distribution, since the control is decentralized and the agents' interactions are asynchronous. Finally, cooperation and learning mechanism contribute to the effectiveness of the optimization.

The two presented approaches have been applied successfully for solving the long-term car pooling problem. Experiments have been performed to confirm the efficiency and the effectiveness of the two approaches. Comparisons among the GGA, the AGA and the CAC approaches have been conducted. For most of the instances, the AGA can provide the best solution quality, however a good hardware environment is required. The GGA is able to provide remarkable solution quality for the instances with random user distribution and the real-world instances. As mentioned in the previous chapter, the CAC approach shows its superiority in processing the instances with clustered user distribution. Therefore, the selection of the algorithm in the real-world implementation depends on the hardware environment of the car pool program provider and the location distribution of the participants. Nevertheless, all three approaches can be considered to be effective and applicable in the real-world implementation.

In the next chapter, we extend our work to the daily car pooling problem. A new daily car pool model which includes multiple destinations in one program will be introduced, and a hybrid resolution method to the problem will be presented.

CHAPTER 5

Extension: Multi-destination Daily Car Pooling Problem

CONTENTS

Abstract

This chapter addresses to an extension problem, the multi-destination car pooling problem. The mathematical model of the problem will be presented at first. Then a Hybrid Ant Colony approach for solving the problem will be outlined. Experimental results are examined in order to evaluate the performance of the resolution method.

5.1 Introduction

Along with our research on the long-term car pooling problem, we also try to make some efforts in improving the daily car pooling problem (DCPP). As introduced in chapter 1, in the DCPP, a number of users declare their availability for picking up or bringing back other users on one particular day. Hence, these users are considered as servers, and the other users being picked up or bringing back are considered as clients. Then the problem becomes to assign clients to servers and to identify the routes to be driven by the servers. According to the definition of the daily car pooling problem, users in a problem are required to go to a common destination. Thus, in the real-world application, the organizer usually separates the users going to different destinations into different car pool projects. Thus, in order to schedule the users with the current daily car pooling system, it is necessary to divide users according to their destinations, and each set of users going to the same destination are considered as an individual instance.

Observed from the real-world application of the daily car pooling, we found that lots of servers travel through their neighbors' destinations before reaching their own ones with available car capacity. However, these servers are not allowed to pick up their neighbors because the neighbors go to different destinations. Dividing users into different car pooling instances based on the destinations results that some instances may have redundant servers, while the other instances may lack of servers.

This situation greatly decreases the effectiveness of serving the clients and potentially increased the travel cost of all the users in the daily car pool project, since if a server can pick up and deliver the clients who go to the destinations other than the server's own, the total travel cost can be greatly saved. Thus, a daily car pool model which includes multiple destinations in one program is required in the real-world application.

In order to respond to this need, a multi-destination daily car pooling model is defined in this chapter. In this model, the server can pick up and deliver clients who go to different destinations as long as the server can accept the length of the detour he/she has to make. A concept called "transfer point" is also defined in this model, which means two car pool servers can meet at an intersection point, where the clients can change vehicles in order to decrease the length of the detour the servers have to make.

So in the Multi-destination Daily Car Pooling Problem (MDCPP), a number of users declare their availability for picking up or bringing back other users on one particular day. These users are considered as servers, then the other users are assigned to servers as clients and the routes to be driven by the servers are identified. In each car pool group, the server and the clients can have different destinations, and each client can be served by two servers during the transition to the destination. Figure 5.1 shows the three situations of picking up and delivery considered in our model: (a) a server

picks up clients, and then they go to a common destination; (b) a server picks up clients and then leads them to their destinations before going to the server's own destination; (c) a server picks up clients and delivers them to a transfer point, and then another server picks them up from this point and delivers them to their destination.

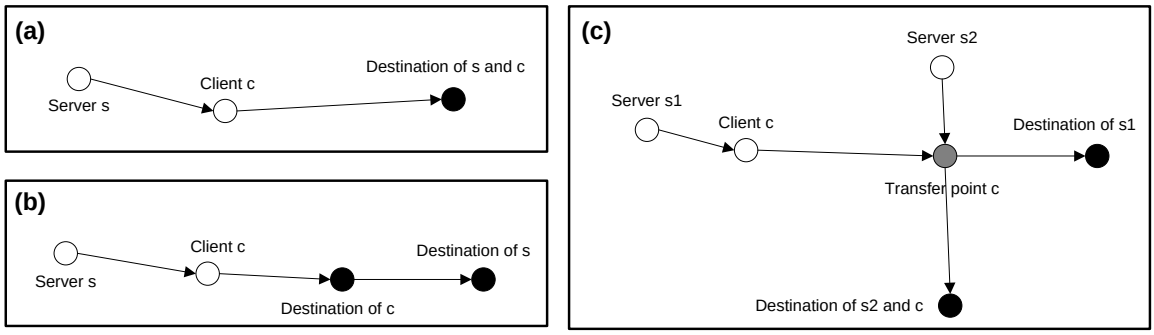


Figure 5.1: Situations of pickup and delivery in Multi-destination Daily Car Pooling.

Observed from figure 5.1, it is clear that the multiple destinations concept is only applicable on daily basis. It cannot be adapted to the long-term car pooling, because each user has to act as a server in the long-term car pooling, and client *c* is not able to pick up and deliver other car pool members within his/her maximum driving time in situation (b) and (c).

Based on the new mathematical model defined for MDCPP, an efficient and effective metaheuristic is developed to solve this problem in the real-world application. Our Hybrid Ant Colony Algorithm (HAC) is based on the Clustering Ant Colony Algorithm introduced in chapter 3; however, we introduce new definitions to the concepts of preference and attractiveness. Moreover, a Transfer Point Searching heuristic (TPS) and a local search procedure are integrated into the algorithm in order to identify the transfer points and to further optimize the best solutions obtained during iterations. Computational results are reported to illustrate the effectiveness of our approach in solving MDCPP.

This chapter proceeds as follows. Section 5.2 describes the MDCPP and its mathematical model. Section 5.3 presents the HAC algorithm for MDCPP. Then, Section 5.4 illustrates the computational results obtained by our method. The last section gives conclusions and perspectives.

5.2 Problem definitions and formulation

The mathematical formulation of the MDCPP will be presented in this section. The objective function and constraints to describe the MDCPP will be introduced in detail manner.

5.2.1 Mathematical model

The MDCPP can be modeled by means of a directed graph $G = (U \cup D, A)$ with $n+o$ vertices, where set $U = \{1, ..., m\}$ is the set of vertices corresponding to the users' home, set $D = \{m+1, ..., n\}$ is the set of vertices associated with the destinations, and $A = \{arc(i, j) / i, j \in U \cup D\}$ is the set of arcs where each $arc(i, j) \in A$ is associated with a positive travel cost $cost_{ij}$ and a travel time t_{ij}. The set U is furthermore partitioned as $U = U_s \cup U_c$, where $U_s = \{1, ..., m_s\}$ is the subset of vertices associated with servers and $U_c = \{m_s+1, ..., m\}$ is the subset of vertices associated with the clients. Let set $O = \{o(i, j, m, n) / i, j, m, n \in U \cup D\}$ be the set of intersection points where $o(i, j, m, n)$ is the intersection point of $arc(i, j)$ and $arc(m, n)$. These intersection points are considered as potential transfer points. Each server $s \in U_s$ enlisted in the multi-destination daily car pooling specifies the car capacity Q_s and a maximal driving time T_s that the server is willing to accept when picking up clients. Each user $i \in U$ has to specify a destination $d_i \in D$, the earlist time e_i for leaving home and the latest time r_i for arriving at destination.

The MDCPP is a multi-objective problem, requiring minimizing the total travel cost of all servers and the amount of unserved clients. Note that the clients can be left on the transfer point and wait for the next server. This results that the waiting time of a client between served by two servers becomes a very important factor to maintain client's satisfaction. Therefore, minimizing the waiting time of all clients is also an objective in MDCPP. In spite of the multiple objectives, it is possible to combine them into a single objective function by still using a penalty concept as in the LTCPP. We define a penalty p_i representing the contribution to the total cost in case a client is not picked up or a server does not pick up any client and a penalty q_i indicating the clients who have to wait in the transfer point. The objectives of MDCPP then can be transformed into an integrated formulation presented as follows.

Define a *pool k* of users and let user *s* be the server and *k* be the amount of members in this pool. Server *s* of pool *k* will use his/her car to pick up the other pool members and then deliver them before going to his/her own destination. The driver thus has to find a Hamiltonian path starts at his/her home, and then passes through all his/her pool members' homes and destinations or transfer points exactly once then ends at his/her own destination. Let $ham(s,k)$ be the shortest above mentioned path, starting from $s \in pool\ k$, connecting all pool members' home $j \in pool\ k \setminus \{s\}$ and transfer points or destinations of all pool members who do not go to the same destination as server *s* and ending in the destination of server *s*, with the constraint that destinations and transfer points must be visited after the corresponding clients. The cost for a server driving directly from his/her home to his/her destination is denoted by $cost_{sd}$, while p_s is a penalty value incurred when the server travels alone, and q_j is the penalty for making the pool member *j* waiting on the transfer point after getting off the car, calculated based on the length of the waiting time wt_j. Then, the cost of pool *k* is defined to be:

$$cost_{pool}(k) = \begin{cases} cost(ham(s,k)) + \sum_{j \in k \setminus \{s\}} q_j wt_j, & if\ |k| > 1, \\ cost_{sd} + p_s, & otherwise. \end{cases} \quad (5.1)$$

The cost for an unserved client *c* is defined as equation 5.2, where p_c is the penalty value for client *c* being not served.

$$cost_{unserved}(c) = p_c \quad (5.2)$$

The total cost of a complete solution of the MDCPP is then defined to be the sum of the costs of the pools in it plus the sum of the costs of the unserved clients. This view optimizes at the same time the three objectives. In our mathematical model, the penalty of a user driving alone is set to be much higher than the cost when he/she drives directly from his/her home to the destination, so it is always more convenient to pool users together than to leave them alone. And because of the penalty for waiting at transfer point, if the client has to wait too long time at the transfer point, it is better to deliver him by the current server instead of changing vehicle.

The transfer points are designed to decrease the travel distance of the servers. Although the transfer points can save the total travel costs, they also increase the inconvenience of the servers and the clients. Moreover, it is very hard to implement the transfer point accurately in the real world application since the real world paths are always not direct lines. Based on the abovementioned reasons, in our model, we limit the amount of transfer point for each client to at most one.

MDCPP being NP can be easily proven since it is transformed from the DCPP and the DCPP is a NP-hard problem. Our MDCPP is difference from the DARP since paths start and end at different locations and the existence of the transfer point mechanism.

5.2.2 Objective function

The problem can be translated in a four indices formulation considering:

- Decision variables:

x_{ij}^{s}: Binary variable equals to 1 if and only if *arc(i,j)* is traveled by server *s*;

y_i: Binary variable equals to 1 if client *i* is not picked up by any server or server *i* does not pick up any client;

ρ_i^s: Binary variable equals to 1 if and only if client *i* is delivered to his/her destination by server *s*;

ψ_{mn}^{is}: Binary variable equals to 1 if and only if client *i* is left on transfer point of *arc(m,n)* by server *s*;

σ_{mn}^{is}: Binary variable equals to 1 if and only if client *i* is picked up on transfer point of *arc(m,n)* by server *s*;
S_i: Positive variable denoting the pickup time of client *i* or the departure time of server *i*.
F_i^d: Positive variable denoting the arrival time of user *i* at a destination *d*;
L_i: Positive variable denoting the arrival time of client *i* on his/her transfer point;
H_i: Positive variable denoting the pick-up time of client *i* on his/her transfer point;
Q_i^k: Positive variable denoting the amount of people in the car of pool *k* after visiting a user *i* or a destination *i* or a transfer point *i*;

- Constants:

ϕ_{id}: Binary value equals to 1 if and only if client *i*'s destination is destination *d*;
$\eta_{(m,n)(p,q)}$: Binary value equals to 1 if and only if there is an intersection between *arc(m,n)* and *arc(p,q)*;
c_{ij}: Positive value denoting the travel cost between users *i* and *j*;
t_{ij}: Positive value denoting the travel time between users *i* and *j*;
Q_s: Positive value denoting the car capacity of server *s*;
T_s: Positive value indicating the maximum driving time specified by server *s*;
e_i: Positive value indicating the earliest time for leaving home of user *i*;
r_i: Positive value indicating the latest time for arriving at the destination of user *i*;
p_i: Positive value indicating the penalty for drive alone server *i* or unserved client *i*;
q_i: Positive value indicating the penalty for client *i* waiting on transfer point;
U_s: Index set of all servers;
U_c: Index set of all clients;
U: Index set of all users;
A: Index set of all arcs;
D: Index set of all destinations;
O: Index set of all intersections.

Objective function:

$$f_{MCPP} = \min\left(\sum_{s\in U_s}\sum_{(i,j)\in A} c_{ij}x_{ij}^s + \sum_{i\in U} p_i y_i + \sum_{i\in U_c} q_i(H_i - L_i)\ \right) \tag{5.3}$$

$$\sum_{j\in U_c} x_{ij}^s = \sum_{j\in U_c} x_{ji}^s \qquad i\in U_c, s\in U_s \tag{5.4}$$

$$\sum_{i\in U_c} x_{di}^s \le 1 \qquad d\in D, s\in U_s \tag{5.5}$$

$$\sum_{i\in U_c} x_{id}^s \le 1 \qquad d\in D, s\in U_s \tag{5.6}$$

$$\sum_{d\in D}\left(\phi_{sd}\sum_{i\in U_c}x_{di}^{s}\right)=0 \qquad s\in U_s \tag{5.7}$$

$$\sum_{d\in D}\left(\phi_{sd}\sum_{i\in U_c}x_{id}^{s}\right)=1 \qquad s\in U_s \tag{5.8}$$

$$\sum_{j\in U_c\cup D}x_{ij}^{s}=\rho_i^{s}+\sum_{m,n\in U_c\cup D}\psi_{mn}^{is} \qquad i\in U_c, s\in U_s \tag{5.9}$$

$$\sum_{m,n\in U_c\cup D}\sigma_{mn}^{is}\le\rho_i^{s} \qquad i\in U_c, s\in U_s \tag{5.10}$$

$$\sum_{s\in U_s}\sum_{m,n\in U\cup D}\psi_{mn}^{is}=\sum_{k\in U_s}\sum_{p,q\in U\cup D}\sigma_{pq}^{ik} \qquad i\in U_c \tag{5.11}$$

$$\sum_{d\in D}\left(\phi_{id}\sum_{j\in U_c\cup D}x_{jd}^{s}\right)\ge\rho_i^{s} \qquad i\in U_c, s\in U_s \tag{5.12}$$

$$(1-\phi_{sd})\sum_{i\in U_c}x_{id}^{s}=(1-\phi_{sd})\sum_{i\in U_c}x_{di}^{s} \qquad d\in D, s\in U_s \tag{5.13}$$

$$x_{mn}^{s}\ge\psi_{mn}^{is} \qquad m,n\in U\cup D, s\in U_s, i\in U_c \tag{5.14}$$

$$x_{mn}^{s}\ge\sigma_{mn}^{is} \qquad m,n\in U\cup D, s\in U_s, i\in U_c \tag{5.15}$$

$$\sum_{s\in U_s}\sum_{m,n\in U\cup D}\psi_{mn}^{is}\le 1 \qquad i\in U_c \tag{5.16}$$

$$\sum_{s\in U_s}\sum_{m,n\in U\cup D}\sigma_{mn}^{is}\le 1 \qquad i\in U_c \tag{5.17}$$

$$\eta_{(m,n),(p,q)}\left(\psi_{mn}^{is}\right)=\sigma_{pq}^{ik} \qquad i\in U_c, m,n,p,q\in U\cup D, s,k\in U_s \tag{5.18}$$

$$Q_i^{s}\le Q_s \qquad s\in U_s, i\in U\cup D\cup O \tag{5.19}$$

$$\sum_{s\in U_s}\sum_{j\in U_c\cup D}x_{ij}^{s}+y_i=1 \qquad i\in U_c \tag{5.20}$$

$$\sum_{(i,j)\in A}x_{ij}^{s}t_{ij}\le T_s \qquad s\in U_s \tag{5.21}$$

$$S_i\ge e_i \qquad i\in U \tag{5.22}$$

$$S_j - S_i \geq t_{ij} - M\left(1 - \sum_{s \in U_s} x_{ij}^s\right) \quad (i.j) \in A \tag{5.23}$$

$$S_i + t_{id} \leq F_i^d \leq r_i \quad i \in U, d \in D \tag{5.24}$$

$$S_m - M(1-\psi_{mn}^{is}) \leq L_i \leq S_m + t_{mn} + M(1-\psi_{mn}^{is}) \quad i \in U_c\ m \in U, n \in U \cup D, s \in U_s \tag{5.25}$$

$$F_s^m - M(1-\psi_{mn}^{is}) \leq L_i \leq F_s^m + t_{mn} + M(1-\psi_{mn}^{is}) \quad i \in U_c\ m \in D, n \in U \cup D, s \in U_s \tag{5.26}$$

$$S_p - M(1-\sigma_{pq}^{ik}) \leq H_i \leq S_p + t_{pq} + M(1-\sigma_{pq}^{ik}) \quad i \in U_c\ p \in U, q \in U \cup D, k \in U_s \tag{5.27}$$

$$F_k^p - M(1-\sigma_{pq}^{ik}) \leq H_i \leq F_k^p + t_{pq} + M(1-\sigma_{pq}^{ik}) \quad i \in U_c\ p \in D, q \in U \cup D, k \in U_s \tag{5.28}$$

$$H_i \geq L_i \quad i \in U_c \tag{5.29}$$

$$x_{ij}^s \in \{0,1\} \quad s \in U_s, (i,j) \in A \tag{5.30}$$

$$y_i \in \{0,1\} \quad i \in U_c \tag{5.31}$$

$$\rho_i^s \in \{0,1\} \quad i \in U_c, s \in U_s \tag{5.32}$$

$$\psi_{mn}^{is} \in \{0,1\} \quad i \in U_c, m, n \in U \cup D, s \in U_s \tag{5.33}$$

$$\sigma_{pq}^{ik} \in \{0,1\} \quad i \in U_c, p, q \in U \cup D, k \in U_s \tag{5.34}$$

$$S_i \geq 0 \quad i \in U_c \tag{5.35}$$

$$F_i \geq 0 \quad i \in U_c \tag{5.36}$$

$$L_i \geq 0 \quad i \in U_c \tag{5.37}$$

$$H_i \geq 0 \quad i \in U_c \tag{5.38}$$

Equation (5.4) is continuity constraint for visiting clients. Equation (5.5) forces the server to go from each destination at most once, while equation (5.6) restricts the server to go to each destination at most once. Equation (5.7) forces the server's path ends at server's destination. Equation (5.8) makes sure each server's destination must be visited. Equation (5.9) and (5.10) force each picked up client must be delivered to the

client's destination. Equation (5.11) makes sure a client left on a transfer point must be picked up eventually. Equation (5.12) restricts the client's destination must be visited if the client is served by a server. Equation (5.13) is continuity constraint for visiting the destinations. Equation (5.14) restricts the transfer point to leave a client must on the path of a server, while equation (5.15) confirms the transfer point to pick up a client must on the path of a server, respectively. Equation (5.16) forces a client can be left on the transfer point at most once, and equation (5.17) restricts a client can be pickup at the transfer point at most once, respectively. Equation (5.18) confirms there must be an intersection between the paths of leaving and picking up a client. Equation (5.19) is car capacity constraint. Equation (5.20) forces each user to be served by only one server or to be penalized, while equation (5.21) is maximum driving time constraints, respectively. Equations (5.22) and (5.23), where M is a big constant, collectively set feasible pick-up times, while equation (5.24) sets minimum and maximum values of feasible arrival times, respectively. Equations (5.25) and (5.26), where M is a big constant, collectively set feasible time to leave a client on a transfer point, while equations (5.27) to (5.29) set feasible time to pick up a client from a transfer point, respectively. Constraints (5.30) to (5.34) are the binary constraints, and constraints (5.35) to (5.38) are the positivity constraints.

5.3 Hybrid ant colony algorithm for the MDCPP

In this section, we explain in detail the concepts and the structure of our Hybrid Ant Colony algorithm (HAC) for the MDCPP. The adaptation of the different components for the MDCPP is described and examined.

5.3.1 Main structure

Our HAC approach for solving MDCPP is based on the Ant Colony Optimization paradigm; the approach tries to assign clients to servers during the ants making their tour. The HAC consists in four components, a pre-sorting process, an ant colony optimization based metaheuristic, a transfer point searching heuristic for determining the transfer points and a local search for further optimizing the solutions. The ultimate goal of this approach is to solve the MDCPP efficiently and obtain a good solution with limited exploration to the search space.

The pre-sorting procedure is designed to partition servers and clients according to several constraints defined based on the servers' convenience, in order to aid and facilitate the future approaches. In this procedure, the servers will be divided into two subsets based on their availability to pass other destinations before going to their own

ones: Servers who are able to visit other destinations; and servers who can only go to their own destinations. The two subsets are used as the start points to build car pools for the ants in the ant colony optimization based metaheuristic. In the same manner, the clients will also be grouped according to their destinations, in order to narrow down the candidates for the ants to assign to a server.

On the basis of the pre-sorting procedure, the ant colony based metaheuristic is then applied. The approach is based on the Clustering Ant Colony Algorithm presented in chapter 3, but its concepts of preference information and attractiveness are redefined. The preference information is designed as the tendencies of assigning a client to a server, while the attractiveness between a client and a server is defined according to the linear distance between the client and the straight line connecting the server and the destinations he/she has to visit.

The new preference and attractiveness information are used to guide the client assignment behavior of the ants, in order to achieve the car pool construction. The first step of the ant colony based metaheuristic is to initialize the preference information. Then, when the ant starts to construct a solution, it will first select a server from the server subsets as the start point to start a new car pool, and then if the car capacity and time window constraints are satisfied, it will continue searching for clients, whose destinations are available to the server, to assign to the server. The probability for the ant to select a client to visit and assign him to a server is based on the preference information and the attractiveness value between the client and the server. During the construction of the car pool, if pooling a client violates the time window constraints, the ant will try to select another client. If the maximum number of times of selection is exceeded, the ant will end the car pool, and stochastically select another server as a new start point of a new car pool.

After a solution has been constructed by an ant, the Transfer Point Searching heuristic (TPS) is applied to create transfer points for car pools. The heuristic proceeds as follows: suppose server *s* makes a relatively long detour to visit client *i*'s destination, the heuristic will then try to locate another server *k* that is going to the same destination as client *i*, and create a transfer point for server *s* to transfer client *i* to server *k*, so the total travel cost can be decreased.

When all ants finish their tours, several best solutions are selected based on the objective function. A local search is then applied to further optimize these selected solutions. The main structure of the local search consists in a loop applying sequentially two operators, and it stops when no improvement made during *x* loops.

At the end of iteration, the composition of the best solutions will be used to update the preference information, and the ancient preference values will decrease with an evaporate rate, in order to enlarge the influence of the new preference information obtained in the current iteration.

The general structure of the HAC is specified as following Algorithm 5.1.

Algorithm 5.1*: Hybrid ant colony algorithm.*

/ Pre-sorting process */*
Partition the servers according to their ability to pass to other destinations;
Partition the clients according to their destinations;

/ Ant colony optimization based metaheuristic */*
Initialize preference and attractiveness.
While *the stop criteria are not met*
  For *k =1, k ≤ the number of ants* ***do***
    Repeat
      Select start point for ants based on the partition of servers;
      Assign clients to servers based on preference, attractiveness and the availability of the server for the clients' destinations;
    Until *all servers are selected;*
    Apply TPS to decide the transfer points;
  End for
  Select the best m solutions;
  Apply local search to the selected solutions;
  Update the preference based on the composition of selected solutions;
End while

5.3.2 Solution representation

The aim to design a representation for the solution is to build a suitable mapping between our problem and the solution generated by the algorithm. Although both direct and indirect coding have been proven to be applicable for the representation of vehicle routing related problem, we favor to select the direct coding for the MDCPP, since the time-consuming encoding and decoding phase of the indirect coding can be avoided. Similar to the Long-term Car Pooling Problem, the representation is designed with two layers. The first layer presents the match between servers and clients, as well as the pickup order of the clients and the visit orders of destinations and transfer points. The second layer records the departure time of the servers, the pickup time of clients and their arrival time at the destinations or the transfer points.

Therefore, the first layer of the proposed representation consists in a set of clusters $Rep = \{R_1, R_2, ..., R_m\}$, and each cluster $R_i = \{S_j, C_k, ... , C_p, L_i, D_s\}$ is a ordered sequence, started with the server of this cluster S_j and followed with clients C_k or transfer point (L_i indicates the transfer points to leave the clients on, P_i refers the transfer points to pick up clients from) or destinations D_s based on the order they are visited by this server. Note that the representation of each cluster may have different length, since the length is based on the number of clients and destinations visited by

the server.

Then in the second layer, for each server S_j and client C_k in each cluster, the departure time T_j or pickup time T_k are associated. For each destination D_s, a value arv_s indicating the arrival time of the server at this destination and a set off_s including the clients who get off the vehicle at this destination are associated.

Note that some clients may correspond to a transfer point; the detail representation concerning this situation will be introduced in section 5.3.6. This representation provides all the information of a user in the Multi-destination Daily Car Pooling Problem. It offers and contributes in a clear manner to design multi-destination daily car pooling problem solutions. An example of the solution representation is shown in figure 5.2.

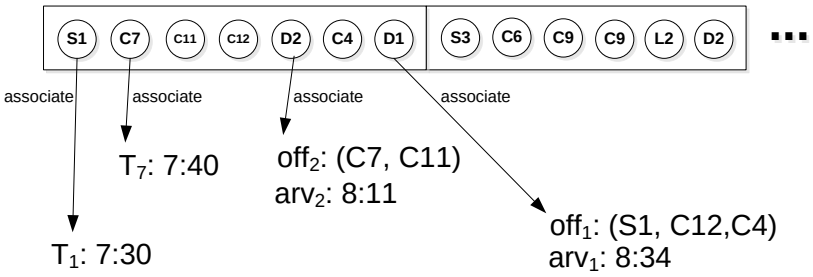


Figure 5.2: An example of the solution representation.

5.3.3 Pre-sorting

Pre-sorting categorizes both servers and clients, it is designed as a preparation procedure for the ant colony optimization based metaheuristic. Two constraints are defined to aid the categorizations of servers and clients.

The servers are divided into two subsets, the M-server subset which contains the servers who are able to visit more than one destination within their maximum driving time, and the S-server subset which consists in the servers who can only go to their own destinations. The selection of the members of each subset is performed by constraints (5.39) and (5.40). The evaluation takes place between each server and each destination except the server's own one.

$$dis_{sn}+dis_{nm} < \quad z \times dis_{sm} \tag{5.39}$$

$$t_{sn}+t_{nm} \; < \; T_s \tag{5.40}$$

where dis_{sm} is the distance between server s and his/her own destination m, t_{sn} is the travel time between server s and another destination n, t_{nm} is the travel time between destination n and destination m, z is a parameter set to adjust the maximum detour length the server can afford, and T_s is the maximum driving time a server willing to accept, respectively.

Constraint (5.39) restricts that the length of a server's detour cannot be longer than *z* times of the distance he/she travels to his/her own destination. Constraint (5.40) confirms that travel to destination *n* will not exceed the server *s*'s maximum driving time. If server *s* is able to satisfy the two constraints when he/she travels to any destination other than his/her own, he/she will be categorized into the M-server subset. Otherwise, the server will be put in the S-server subset. For each processed server, his/her available destinations will be recorded with him, shown in figure 5.3.

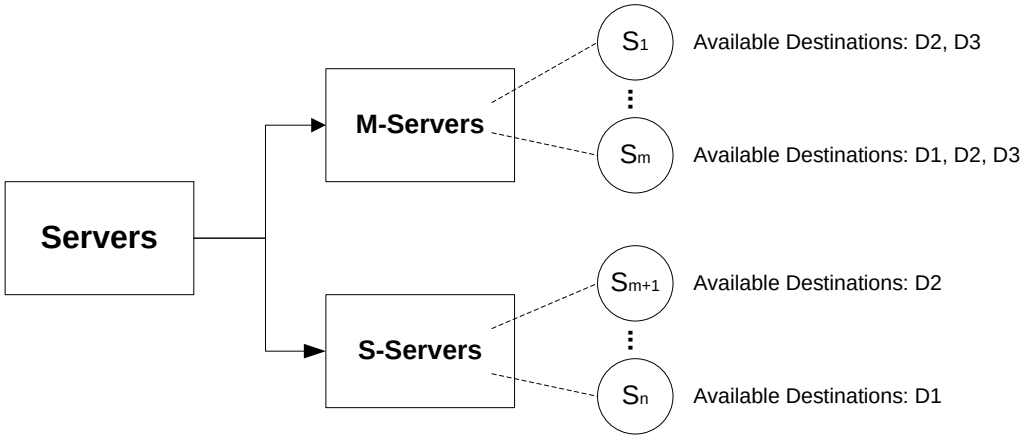


Figure 5.3: Categorization of servers in pre-sorting procedure.

These two subsets will be both considered as the start points for building car pools for the ants in the ant colony optimization based metaheuristic, but they are given different priorities when ants process them. The servers in the M-server subset must be processed before the ones in the S-server subset, only after all servers in the M-server subset have been assigned with car pool members, the ants start to process the servers in the S-server subset. By defining this concept, we achieve the favor to the servers who can travel to multiple destinations, as they can serve more clients than the servers who go to only their own destinations.

The clients are also divided into several subsets; the number of subsets is based on the number of destinations of an instance. The clients going to the same destinations are organized into the same subset. In the ant colony optimization based metaheuristic, when an ant searches clients to assign to a server, only the clients going to the available destinations of the server are visible to the ant. That is, only the clients with the destinations the server can reach without violating the constraints (5.39) and (5.40) are candidates for the ant to select. By applying this mechanism, the amount of candidates for an ant has been significantly decreased.

5.3.4 Preference information

The Clustering Ant Colony structure is kept in our ant colony optimization based metaheuristic, but the concepts of preference and attractiveness are redefined. The preference information in our metaheuristic is distinct from the Clustering Ant Colony Algorithm, because it is defined between each server and each client. The preference information is stored in an $m \times n$ matrix where m is the number of servers and n is the number of clients in an instance. The preference values of the matrix indicate the tendency level to assign a client to a server, as shown in figure 5.4. These values are experiences gained from the best solutions obtained in each iteration, and are used to guide the ant for constructing car pools in future iterations.

Preference matrix in HAC

	C_1	C_2	C_3	C_4	...	C_n
S_1	*0*	*0.5*	*1.1*	*0*	*0.5*	*0*
S_2	*0.5*	*0*	*1.4*	*0.7*	*0*	*0*
⋮	*1.0*	*0.2*	*0*	*0*	*0*	*0*
S_m	*0*	*0.7*	*0*	*0*	*0.8*	*1.2*

Figure 5.4: The preference information matrix in HAC.

In the initialization of the preference information, several time window constraints are pre-checked between the servers and the clients whose destinations are available for the servers. The preference values between the server and the clients whose destinations are not feasible for the servers are set to be zero. Let d_s be the destination of server s, d_i be the destination of client i, t_{ij} be the travel time between two locations, respectively. Constraints (5.41) and (5.42) examine whether server s and client i can both be able to arrive on time, if server s picks up client i before going to the destination. Constraint (5.43) checks if the pickup time of client i is too late for server s to arrive at the destination on time. Note that, if client i has a different destination from server s, the server must deliver the client first, and then go to his/her own destination. If pooling server s and client i together cannot satisfy the abovementioned constraints, the preference value w_{si} is set to zero, which indicates there is no probability that client i will be assigned to server s by ants. By applying this procedure, we are able to remove some car pool combinations which do not belong to any feasible solution, so the computing time for the ants to search for car pool members is further decreased.

$$e_s + t_{si} + t_{id_i} + t_{d_i d_s} \leq r_s \tag{5.41}$$

$$e_s + t_{si} + t_{id_i} \le r_i \tag{5.42}$$

$$e_i + t_{id_i} + t_{d_i d_s} \le r_s \tag{5.43}$$

Then, if the constraints are satisfied, the preference values between each server and each client are initialized by the geographic distance difference between server *s* going to his/her destination directly and server *s* picking up and delivering the client *i* before going to his/her own destination, and their earliest departure time difference, shown as equation (5.44).

$$w_{si} = \alpha \times \frac{1}{dis_{si} + dis_{in} + dis_{nm} - dis_{sm}} + \beta \times \frac{1}{\left| e_s + t_{si} - e_i \right|} \tag{5.44}$$

where dis_{si}, dis_{in}, dis_{nm} and dis_{sm} are the geographical distances between server *s* and client *i*, between client *i* and *i*'s destination *n*, between client *i*'s destination *n* and server *s*'s destination *m*, and between server *s* and *s*'s destination *m*, respectively. t_{si} are the travel time between server *s* and client *i*. e_s and e_i are the earliest departure time of server *s* and client *i*. α and β are weight factors. Equation (5.44) indicates that the shorter the detour the server *s* has to make to pick up client *i*, the higher the initial preference between them. If the arrival time of server *s* at client *i*'s home is close to client *i*'s earliest departure time, they will also obtain a higher preference value between them. Note that the constants in the abovementioned equation have both distance unit and time unit, so the factors α and β are designed to adjust these values. Similar to the Clustering Ant Colony Algorithm, the two denominators in equation (5.44) are limited to a minimum value σ, which replaces the denominators if their values are less than σ. This mechanism is designed to avoid generating too large preference values.

In each iteration, an ant starts its tour from a server *s*, selected stochastically from the M-server set obtained in pre-sorting procedure. Then the ant tries to assign clients to server *s*. The probability for the ant to select a client *i* to visit and assign to server *s* is based on the preference information and attractiveness between client *i* and server *s*. The attractiveness will be introduced in detail in next section. The probability for the ant to select a client *i* is performed by a roulette wheel selection procedure, as shown in equation (5.45). As abovementioned, only the clients with the destinations can be visited by server *s* without violating the constraints have positive preference values, which indicates they are visible for the ant to select.

$$\text{Pro}bability_{si} = \frac{(w_{si})^a (\eta_{si})^b}{\sum_{j \in H} (w_{sj})^a (\eta_{sj})^b} \tag{5.45}$$

where w_{si} is the preference value between server s and client i, while η_{si} is the attractiveness value, respectively. H is the set of visible candidates of server s, who have not been assigned to any car pool yet, while a and b are adjusting parameters, respectively.

If the client selected by the ant can satisfy the time window constraints of the server and the existing clients in the car pool, the selected client will be added into the car pool. Otherwise, the selection is considered as failed and the ant will try to select another client until the maximum number of times of failed selection y is exceeded. When a new client is assigned to the car pool, the new order of pickup and delivery of the clients will be updated. The update is based on the calculation of the attractiveness, which will be presented in section 5.3.5.

When the current constructing car pool reaches its car capacity or exceeds the maximum number of times of failed selection, the car pool will be closed. And then the ant will select stochastically another server from the M-server, or S-server if the servers in M-server subset are all processed, then continue to assign clients to the server, as shown in figure 5.5.

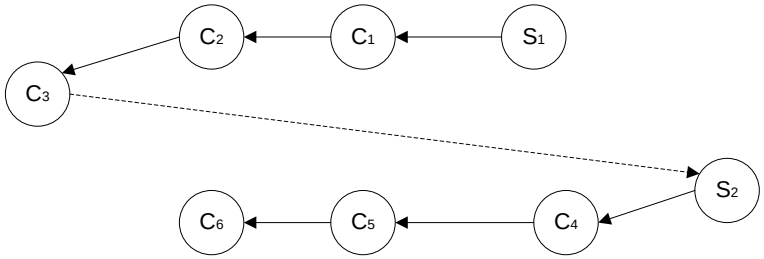


Figure 5.5: The procedure of ant searching for car pool members.

After a solution has been constructed by an ant, the Transfer Point Searching heuristic (TPS) is applied to create the transfer points. The TPS will be presented in detail in section 5.3.6. When the transfer point is built, the waiting time of client i on the transfer point is calculated and associated to him, in order to aid the evaluation at the end of iteration.

When all ants finish their tour, an evaluation is performed to all the solutions generated by the ants; the evaluation is given by equation (5.3), which is the objective function of the MDCPP. After the evaluation, the first m best-fit solutions are selected to be applied with a local search, in order to further optimize the solution.

Then, before updating the preference values with these solutions, all weight values w_{si} in the preference information matrix will decrease with an evaporate rate μ, as shown in equation (5.46), where w''_{si} is the preference value of the previous iteration, in order to enlarge the influence of the new preference information obtained in current iteration. Afterwards, for each selected solution s, the preference values between the server and the clients in the same car pool consist in an augmentation, as shown in

equation (5.47), where w'_{si} is the preference after evaporation, f_{avg} is the average fitness of the whole ant colony, f_k is the fitness of current selected solution s, S is the set of all selected solutions, and factor λ is a weight factor, respectively. For the clients who have been transferred during the travel, only the preference value between the clients and their original servers are updated.

$$w'_{si} = \mu \times w''_{si} \tag{5.46}$$

$$w_{si} = w'_{si} + \sum_{s \in S} (\lambda \times \frac{f_{avg} - f_s}{f_{avg}}) \tag{5.47}$$

Figure 5.6 shows an example of preference matrix updating.

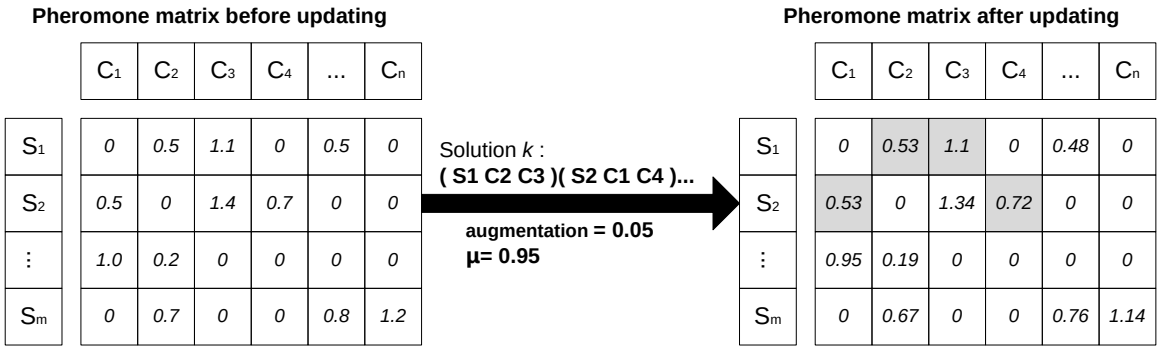


Figure 5.6: Updating the preference matrix in HAC.

5.3.5 Attractiveness

The paradigm of the Ant Colony Optimization (ACO) involves the movement of a colony of ants through the different states influenced by two local decision policies, pheromone and attractiveness. In our ant colony optimization based metaheuristic, the attractiveness is considered as the level a client is attracted by the theoretical shortest path of the server. Therefore, the attractiveness between client *i* and server *s* is defined as the reciprocal of smallest linear distance between client *i* and the straight lines connecting server *s* and the destinations the server has to pass.

During the ant colony optimization based metaheuristic, when the ant is searching client *i* for server *s*, a "standard path" is designed for calculating the attractiveness. The standard path is a Hamilton path starts from server *s* and connects sequentially the destinations the server has to visit to deliver his/her current car pool members as well as client *i*, as shown in figure 5.7. Thus the standard path is modified if client *i* goes to a different destination.

For instance, in figure 5.7, server *s*'s destination is d_s, and the two clients are with destination d_{c1} and d_{c2}. Then, in the beginning of a car pool construction, the standard path is the section of line connecting the server *s* and the server *s*'s destination. When the ant tries to insert client c_1 with different destination into the car pool, the standard path is changed to the Hamilton path starts from the server, passing the destination of c_1 and ends at server *s*'s destination. In the same manner, when the ant tries to insert client c_2 into the car pool, the standard path of this car pool is the Hamilton path starts from server's home connecting d_{c1} and d_{c2}, ends at d_s.

As the standard path is the shortest path to go through the destinations for the server, if the server picks up the clients close to this path, he/she will make only relatively short detour. Thus, the standard path is used as the guide line to calculate the attractiveness for guiding the ants; the concept is designed to ensure that the distribution of selected clients vibrates around the standard path with a narrow width.

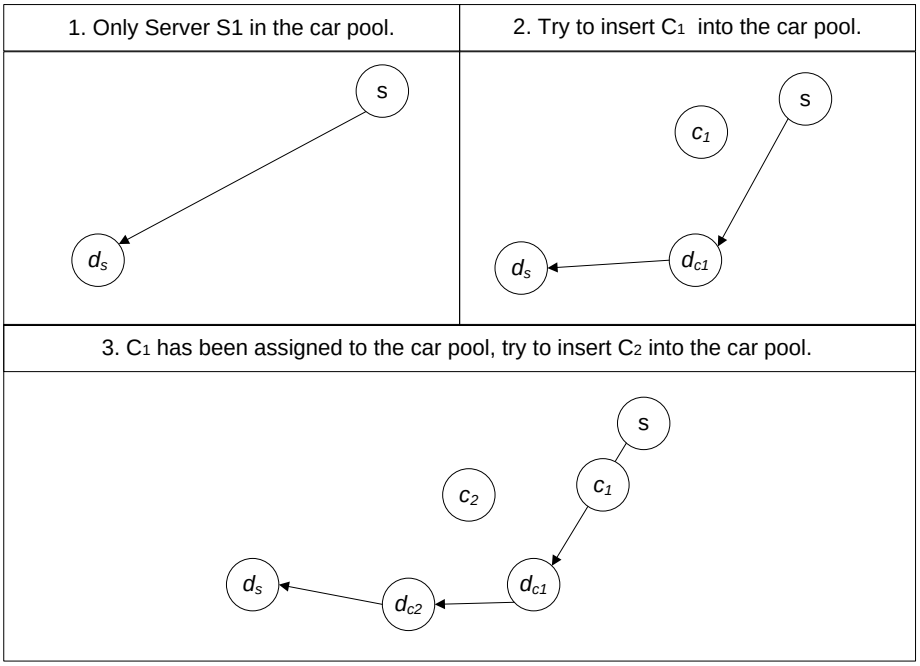


Figure 5.7: The modification of the standard path.

When ant makes its tour, the attractiveness is calculated according to the standard path of the server. We calculate the linear distance between the candidate client *c* and each section of the standard path before reaching *c*'s destination, and then choose the reciprocal of shortest one as the attractiveness value of this client to the server, as shown in figure 5.8. Note that if the projection point of the client is not on the corresponding section of the standard path, an extra distance will be added to the attractiveness distance. The extra distance is calculated as the linear distance between the projection point and the closest end of the section of the standard path. In figure 5.8, server *s*'s destination is d_s, candidate client *c*'s destination is d_c, and d_x is the destination of an existing client in server *s*'s car pool. d_{c_mn} indicates the linear distance from

client *c* to the section between node *m* and node *n*, and *P* refers to the projection points of client *c* on each section of the standard path. In the example, the distances between *c* and the standard paths are d_{c_sdx} and $d_{c\_dxdc}+d_{pc2dx}$. Thus, the attractiveness value between *c* and *s* is calculated as the reciprocal of d_{c_sdx}, since d_{c_sdx} is shorter than $d_{c\_dxdc}+d_{pc2dx}$.

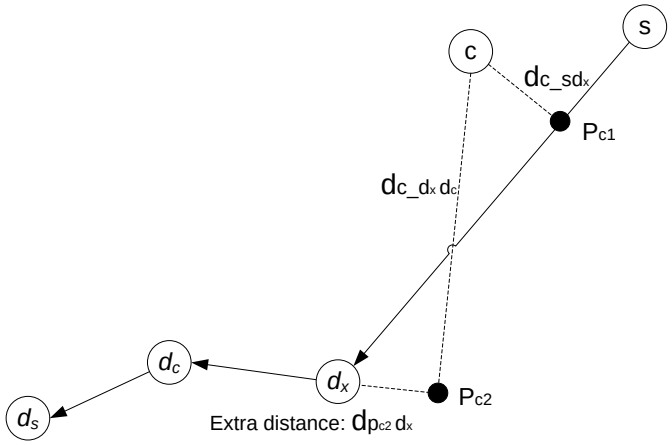


Figure 5.8: The distance between a client and the standard path.

Note that only the distances between client *c* and the sections of the standard path before reaching *c*'s destination can be calculated and used. In figure 5.8, client *c* can only have the projection point on the section sd_x and d_xd_c of the standard path since his/her destination is d_c. The distance between client *c* and the sections d_cd_s of the standard path is not calculated because, even if the distance was the shortest, the server has to travel to an opposite direction to deliver the client. This is usually inconvenient and expensive for the server, thus it is better to leave the client to other servers.

At last, according to the section of the standard path selected to calculate the attractiveness and the location of the corresponding projection point, we are able to obtain the position for a client to be inserted in the server's pickup and delivery sequence. For instance, in figure 5.8, client *c* will be inserted in the pickup and delivery sequence between *s* and d_x, since the attractiveness of *c* is calculated based on the distance to section sd_x.

Therefore, the distance dat_c for calculating the attractiveness is calculated as equation (5.48). By connecting client *c* to the two ends of section *mn* of the standard path, we can obtain a triangle. If $\hat{cmn}$ and $\hat{cnm}$ are acute-angles or right-angles, dat_c simply equals to the distance d_{c_mn} between client *c* and section *mn*. Otherwise, dat_c is added with an extra distance $min(d_{pm}, d_{pn})$.

$$dat_c=\begin{cases} d_{c\_mn} & \text{if } \hat{cmn} \text{ and } \hat{cnm} \le 90^{\circ} \\ d_{c\_mn}+min(d_{pm},d_{pn}) & \text{otherwise} \end{cases} \tag{5.48}$$

where d_{c_mn} indicates the linear distance from client c to the section between m and n, while d_{pm} and d_{pn} indicates the linear distances from projection point p of client c to m and n.

And the attractiveness for client c to a server s will be:

$$\eta_{sc} = \frac{1}{min(dat_c)} \tag{5.49}$$

where dat_c indicates all possible distances from client c to the sections of the standard path.

When a new client is assigned to a car pool, according to its projection point on the standard path, the pickup and delivery sequence will be updated. The client with the projection point closer to the server will be picked up first. The attractiveness is essential in our ant colony optimization based metaheuristic. Its role is not only a factor for ants to select clients to assign to servers, but also to indicate the position where the client should be located in the pickup and delivery sequence.

5.3.6 Transfer point searching heuristic

After a solution has been constructed by an ant, the Transfer Point Searching heuristic (TPS) is applied to create the transfer points. As mentioned in the mathematical model, the transfer point can increase the inconvenience of the servers and the clients. Thus, in our model, we limit the number of transfer points to at most one for each client.

In TPS, we define a parameter to decide whether a detour of a server is considered to be long. In a solution, if server s is confirmed to make a long detour to deliver client i, the heuristic will try to locate another server k that is going to the same destination as client i and has available car capacity, then try to create a transfer point for server s to transfer client i to server k, in order to decrease the total travel cost.

The servers, who travel multiple destinations and have the travel distance is more than w times of the distance from his/her home directly to his/her destination, are selected and put into a list. The list is organized in decrease order of the length of the servers' detours, and the server with longer detour will be processed first. Then, all the servers in the list proceed with the following procedures, as shown in figure 5.9. Suppose s is a server with destination d_s, then for every other destination d_i server s has to visit (in the case of figure 5.9, d_1 is the other destination server has to visit), if there is no existing transfer point on the paths linking d_i (in figure 5.9, the paths are c_2d_1 and d_1d_s) and no client going to d_i has been transferred before, we remove d_i from the original path to construct a conjecture path, and the clients going to d_i are marked to be transferred in the next step. Otherwise, we skip d_i and process the next destination. Then, we select randomly a server who goes through destination d_i and examine the availability to construct a transfer point. The process for destination d_i

continues until a transfer point is created or all possible servers who go through destination d_i are checked.

A server *k* must satisfy the following constraints to create a transfer point:

(1) There must be an intersection between server *k*'s path and server *s*'s conjecture path;

(2) The intersection must be on the part of the conjecture path which is visited after picking up the clients of server *s* who need to be transferred.

(3) Server *k* must have available car capacity to serve the clients transferred from server *s*.

(4) Calculate the coordinates of the transfer point, and then check the arrival time of both servers at the transfer point. The time for server *k* to pick up the clients must be later than the time for server *s* to leave the clients.

The constraints are examined sequentially; the next constraint is examined only if the previous constraint is satisfied. Note that, the coordinates calculated in the fourth constraint are memorized, so the repetitive calculation can be avoided in the future iterations.

Since the verification of the constraints is time consuming, we apply the first improvement policy. That is, the transfer point is confirmed as soon as the total cost of the two car pools decreases. Note that, the path of server *s* will be modified to be same as the conjecture path after the creation of the transfer point.

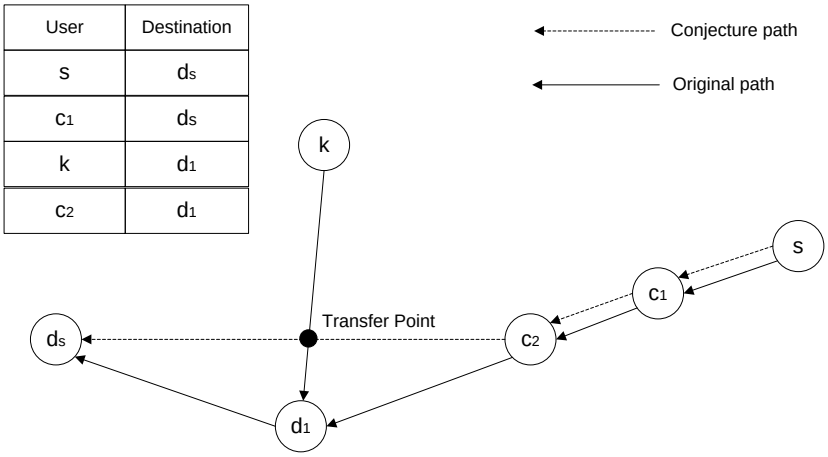


Figure 5.9: Construction of a transfer point.

The representation of the transfer point in a solution is introduced in figure 5.10. The transfer point to leave clients is named with L_i, while the transfer point to pick up the corresponding clients is named with P_i. Each transfer point *i* is associated with set SL_i of clients being left or set SP_i of clients being picked up, the distance *dis* and travel time *tim* between the previous visited node and transfer point *i*, and the time *arv* of the server arriving at this transfer point.

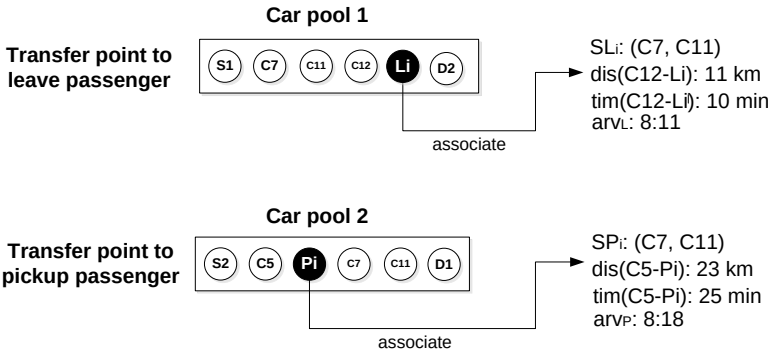


Figure 5.10: Representation of a transfer point.

5.3.7 Local search procedure

As abovementioned, when all ants have finished their tours and the TPS is applied, several best solutions are selected to be improved by a local search procedure. The main structure of local search consists in a loop applying sequentially two operators. The local search stops when no improvement made during *x* loops. The same forbidden list mechanism as presented in chapter 3 is applied to avoid repetitive calculations.

***Algorithm 5.2**: Local search in HAC.*

Local_Search_Operators[] = { Swap, Move }
Create the forbidden list;
Repeat
  For** each operator in Local_Search_Operators **do
    Select car pools for operator;
    For** selected car pools **do
      Check the forbidden list;
        ***If** not in the list, apply the operator;*
          ***If** solution is improved, update the solution;*
          Remove corresponding information of the current car pool from the forbidden list;
          ***Else** record the operation in the forbidden list;*
        ***Else** skip the operation and process next selected car pool;*
    End for
  End for
***Until** the stop criteria of the local search is met.*
Clear the forbidden list.

The two operators are designed specifically based on the characteristic of the MDCPP. The main structures of the operators are similar. For each operator, we first select several car pools according to the selection rules defined by each operator. Then, for each operation performed by an operator, we check the forbidden list. If there is corresponding information in the list, which means we applied the same operation to the same users in the past and made no improvement, we will skip the operation. Otherwise, we apply the operator and see whether there is an improvement obtained. If the solution is improved, we update the solution and remove the corresponding information of the modified car pools from the forbidden list. And if the solution is not improved, which indicates the operation on the selected car pools is not beneficial. We will record this information in the forbidden list, in order to avoid repeating the same operation again to the same car pools.

Swap operator

The operator first stochastically selects *u* servers who make long detours. The servers, who travel multiple destinations and have the travel distance is more than *v* times of the distance from his/her home directly to his/her destination, is considered making a long detour. The length of detour is calculated as the distance difference between the server's travel distance and the distance from server's home directly to his/her destination. The selection is performed by a roulette wheel selection based on the length of the detour made by the server. The probability of server *s* to be selected is calculated by equation (5.50).

$$\text{Probability}_s = \frac{\text{detour}_s}{\sum_{k \in K} \text{detour}_k} \tag{5.50}$$

where $detour_s$ is the length of detour of server *s* and *K* is the set of all servers.

For each selected server *s*, the operator tries to swap every existing car pool member of server *s* with any client *j* whose destination is visited by *s* and has positive preference value to *s*. The move is confirmed as soon as an improvement is obtained.

Note that, the clients being transferred cannot be swapped, in order to avoid affecting the schedule of the car pool where the clients being transferred from or to. An example of the operation of the swap operator is presented in figure 5.11.

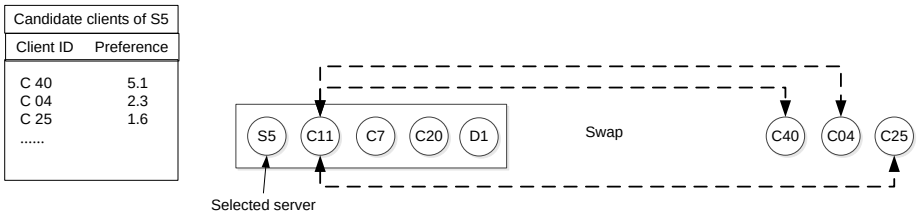


Figure 5.11: An example of the operation of swap operator.

Move operator

The move operator tries to move a client from a selected car pool to a server with available car capacity. It first selects *t* servers who make long detours. The selection is performed by a roulette wheel selection still based on equation (5.50).

For each selected server *s*, the operator attempts to move every car pool member *i* to any server who passes *i*'s destination, and has positive preference value to *i* and available car capacity, as shown in figure 5.12. The move is confirmed as soon as an improvement is obtained. Following the same manner as the swap operator, the clients corresponding to the transfer points cannot be moved.

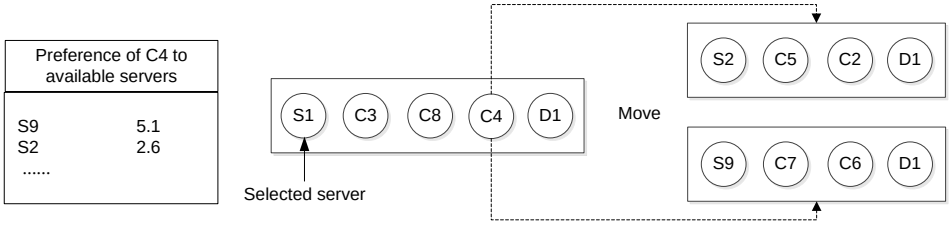


Figure 5.12: An example of the operation of move operator.

5.4 Experimental Results and Analysis

5.4.1 Benchmarks

Computational experiments have been conducted to examine the performance of the proposed approach. Since no literature can be found on MDCPP, we created three new sets of instance based on our LTCPP benchmarks presented in chapter 1. We select a few users from each LTCPP instance to be servers and the rest of the users in the instance are considered as clients. We also define several new destinations, so the LTCPP instances can be conversed to the MDCPP instances.

For the original set C instance of the LTCPP, we first calculate the geographical gravity center of all users in an instance and select 20% of users who are the farthest from the gravity center to be servers, thus the servers are located in the surrounding areas of the user distribution. The destinations are defined by selecting the gravity centers of the rest users who are considered as clients. We divide the clients into clusters with same size, the number of clusters equals to the number of destinations we want to define. Then, for each cluster we calculate the gravity center of each cluster and consider it as a destination. At last, each server and client are randomly assigned with a destination. In the original set R instance, random 20% of the users are selected to be servers and the rest of the users are set to be clients. The destinations are gener-

ated and assigned in the same manner as set C instances. In the original set W instance of LTCPP, which is a real-world instance, we keep the original destination and use other campuses of the Artois University to be the second and third destinations. The selection of servers and clients, as well as the assignment of their destinations, are performed randomly as in the set R instances. At last, minor modifications are done to the new set R and set W instances by swapping the location of the clients, who are obviously impossible to be picked up, with some random selected servers. Figure 5.13 shows a general idea of the user distributions in MDCPP instances.

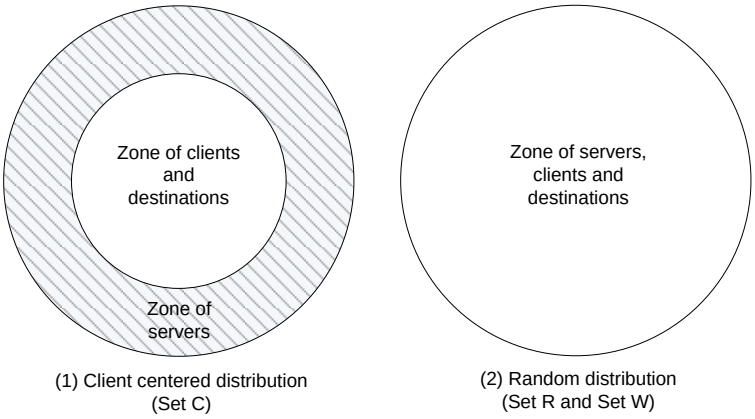


Figure 5.13: Different user distributions in MDCPP instances.

Thus we obtained three sets of instances for MDCPP, still named with C, R and W. Each set includes 9 instances with various sizes from 100 to 400 (565 for set W).

5.4.2 Configuration

Parameter setting for the presented HAC algorithm is specified as follows.

Pre-sorting: $z = 1.5$;

Number of ants: *90*;

Initialization parameters: $\alpha = 0.9$, $\beta = 0.1$, $\sigma = 1$;

Probability parameters: $a = 2$, $b = 1$, $y = 3$;

Preference updating parameters: $\lambda = 0.1$, $\mu = 0.95$, $n = 10$;

Penalty parameters: $p = 2 \times$ *user's direct travel cost to the destination*, $q = 2$;

TPS parameters: $w = 1.3$;

Local search: $u = t = 20$ % *of the amount of all car pools*, $x = 2$, $v = 1.2$.

Given limited computational resources and combinatorial complexity, parameter values were determined empirically over a few intuitively selected combinations, choosing the one that yielded the best average output.

5.4.3 Results obtained with HAC

The HAC algorithm was implemented in JAVA, under Eclipse 6, and all results were obtained running the code on a Windows Operating System with Intel Core i7 740QM 2.9 GHz CPU and 4 GB of RAM. The algorithm has been executed 30 times for each instance, where each execution is given 1000 iterations.

In order to evaluate the gain of solving users with multiple destinations together, in our first set of experiments, we compare the results obtained by our MDCPP model with the ones generated by classic daily car pooling problem (DCPP) model. We divide each instance into sub problems according to the destination of users. Therefore each sub problem includes only one destination, which transforms the instance into a classic DCPP one. Since the DCPP is less complex than the LTCPP and MDCPP, so it is possible to obtain a good solution to an instance of 100 users by using CPLEX with several hours of computing. Thus, we divide each DCPP instance again into smaller ones with around 100 users. At last, the total costs, as well as the traveled alone servers and unserved clients of each sub problem are summed up and compared with the results provided by HAC with the MDCPP model. By the comparison, we are able to examine the benefits of considering an instance as a MDCPP instead of several DCPPs.

In the result tables, *Size* with the format *[number of users (number of servers / number of clients)]*, refers to the amount of users, servers and clients in each instance, respectively. *N* indicates the number of destinations in an instance.

In table 5.1, we compared the experimental results of HAC with the ones generated by the CPLEX approach. The HAC outperforms the CPLEX approach on all instances.

Size	N	Set C instances				Set R instances				Set W instances			
		HAC			CPLEX	HAC			CPLEX	HAC			CPLEX
		Best	Avg	Time(s)	Best	Best	Avg	Time(s)	Best	Best	Avg	Time(s)	Best
100(20/80)	2	1629.0	1641.3	11	2174.2	1963.1	1989.7	12	2372.1	878.7	886.0	10	1082.9
100(20/80)	2	1516.6	1542.7	10	1984.1	2091.4	2097.2	14	2495.2	791.4	798.2	10	1019.5
100(20/80)	2	1703.5	1715	10	2354.4	2139.1	2165.5	12	2763.6	1037.4	1040.0	12	1238.6
200(40/160)	2	2376.2	2413.1	35	3385.2	3408.7	3454.0	38	4735.9	1427.5	1464.0	41	2043.3
200(40/160)	2	2839.0	2883.5	41	4184.8	3997.5	4082.1	43	4822.8	1356.5	1358.2	33	1812.4
200(40/160)	2	3835.6	3879.5	37	5060.4	2781.3	2843.1	42	3950.5	1209.3	1229.5	34	1759.6
400(80/320)	3	4924.7	5049.6	297	7454.8	5829.0	5953.7	344	8077.7	2331.8	2404.5	255	3534.3
400(80/320)	3	3969.2	4091.4	332	6455.3	4722.5	4958.0	271	6395	2979.7	3097.4	321	4145.6
400(80/320)	3	5126.2	5267.2	281	7238.5	6187.0	6298.0	368	8831.5	4435.0	4538.8	318	5781
Total		27920	28483.3	1054	40291.7	33119.6	33841.3	1144	44444.3	16447.3	16816.6	1034	22417.2

Table 5.1: Experimental results of HAC and CPLEX.

Table 5.2 shows the percentage the HAC outperforms the CPLEX approach on the three sets of instances, in the aspect of average solution quality. The computing time is not listed. Since the CPLEX approach costs several hours to generate the result, it is obvious that the HAC is significantly faster. For each instance, the outperforming percentage is calculated as *(CPLEX's value – HAC's value) / CPLEX's value*. Each value in table 5.2 is obtained by averaging the outperforming percentages of the three same-size instances. According to our experiments, the HAC can improve the CPLEX approach's results by 28.1%, 22.4% and 24.2% on three different sets of instances.

Size	Set C instances	Set R instances	Set W instances
100	24.6%	17.9%	18.6%
200	27.7%	23.5%	27.8%
400	32.0%	25.8%	26.2%
Avg	28.1%	22.4%	24.2%

Table 5.2: Solution quality comparison.

According to table 5.1 and 5.2, we believe that, solving the instances as a MDCPP can always provide better solutions than solving the instance as several DCPPs. The total cost is decreased significantly. Therefore, the comparison reveals the effectiveness MDCPP model and the efficiency of the HAC in solving the MDCPP.

Size	N	Set C instances		Set R instances		Set W instances	
		With TPS	No TPS	With TPS	No TPS	With TPS	No TPS
100(20/80)	2	1641.3	1835.4	1989.7	2157.2	886.0	915.0
100(20/80)	2	1542.7	1813.1	2097.2	2252.8	798.2	837.8
100(20/80)	2	1715.0	1905.0	2165.5	2495.0	1040.0	1138.8
200(40/160)	2	2413.1	2921.5	3454.0	4149.1	1464.0	1738.9
200(40/160)	2	2883.5	3386.3	4082.1	4311.7	1358.2	1635.3
200(40/160)	2	3879.5	4504.1	2843.1	3459.5	1229.5	1465.3
400(80/320)	3	5049.6	6051.3	5953.7	6846.3	2404.5	2715.9
400(80/320)	3	4091.4	4716.4	4958.0	5833.7	3097.4	3586.1
400(80/320)	3	5267.2	6406.9	6298.0	7160.9	4538.8	5064.8
Total		28483.3	33540.0	33841.3	38666.1	16816.6	19097.8

Table 5.3: Comparison with the results obtained without TPS.

Size	Set C instances	Set R instances	Set W instances
100	11.8%	9.3%	5.5%
200	15.4%	13.3%	16.3%
400	15.9%	13.4%	11.8%
Avg	14.4%	12.0%	11.2%

Table 5.4: Solution quality comparison with the results obtained without TPS.

Table 5.3 and table 5.4 show the improvement given by the Transfer Points Search heuristic (TPS). The average solution quality of HAC has been compared with the one obtained by disabling the TPS mechanism. According to the tables, all the solutions benefit from the TPS approach. The total cost of all instances has been decreased by this procedure. The improvements vary from the size of the instances. The large size instances, such as 200 and 400, obtain the most significant improvements.

The accuracy of the HAC is examined by calculating the standard error (column Std) of the solutions obtained in 30 runs of each instance. The solution quality difference (column Diff) between the best found solution quality and the average solution quality of *each* instance in the previous tables is also calculated. Table 5.5 shows the *average* of the abovementioned values of the three same-size instances. The average differences between the best found solution and the average solution of the three sets of instances are 1.7%, 1.8% and 1.7%, respectively, which indicates the HAC approach can be considered to be accurate for a metaheuristic.

Size	C set instances				R set instances				W set instances			
	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)
100	1616.4	1633.0	8.5	1.0	2064.5	2084.1	11.3	0.9	902.5	908.1	3.6	0.6
200	3016.9	3058.7	17.2	1.4	3395.8	3459.7	27.9	1.8	1331.1	1350.6	11.7	1.4
400	4673.4	4802.7	65.8	2.7	5579.5	5736.6	93.4	2.7	3248.8	3346.9	76.4	2.9
Avg	3102.2	3164.8	31.8	1.7	3680.0	3760.1	44.2	1.8	1827.5	1868.5	30.6	1.7

Table 5.5: Evaluation of the accuracy of the HAC.

5.5 Conclusion

In this paper we defined a new car pooling problem model, Multi-destination Daily Car Pooling Problem (MDCPP). The mathematical formulation has been introduced in detail manner. Then, we introduced the HAC algorithm, a Hybrid Ant Colony Algorithm to solve the MDCPP. We presented in detail the four components of HAC algorithm, a pre-sorting procedure, an ant colony optimization based metaheuristic, a heuristic process designed inside the ant colony structure for determining the transfer point and a local search for further optimize the solutions.

The presented approach has been applied successfully for solving the MDCPP. The experiments of the HAC have been performed on three sets of structurally different instances. Each set includes large scale instances. The experimental results are compared with a CPLEX based decomposition approach, and the comparison has proven that solving the instances as a MDCPP can provide better solutions than solving the instance as several DCPPs. Experiments also have been performed to confirm the effectiveness of the Transfer Point Searching heuristic. Thus, it has been demonstrated that the HAC algorithm is an effective approach for solving the MDCPP.

Conclusions

This thesis has focused on the development of metaheuristics to solve long-term car pooling problem (LTCPP). The vast amount of research on metaheuristics indicates that they have established themselves as an effective tool to deal with complex optimization problems.

In this thesis, different classes of metaheuristics are presented. Experimental results demonstrate that the methods are effective and efficient for solving large scale LTCPP instances.

The contributions that stem from this PhD thesis are:

• Present a state-of-the-art on car pooling problem. It covers the description of the problem, the mathematical model and its recent solving methods.

• Define three sets of structurally different instances for the long-term car pooling problem.

• Provide an efficient way to solve the LTCPP. This contribution is based on the development of Variable Neighborhood Search (VNS) algorithm. The interest of this algorithm consists in the ability of shifting from a neighborhood to another one throughout the optimization process. This ability offers an effective mechanism for tracking the optimum in the search space. Neighborhoods particularly designed for the LTCPP have been integrated to increase the efficiency of this approach. The experiments demonstrate that the approach is able to provide good solutions within very short computational time.

• Present a high performance swarm intelligence metaheuristic for the LTCPP. This metaheuristic is called Clustering Ant Colony Algorithm (CAC). The approach is based on the ant colony optimization paradigm. A preference concept is defined to give the ants the abilities to cluster users. The approach achieves to merge the clustering and routing operations during the optimization process, thus it possesses the ability to generate high solution quality. The preference mechanism is proven to be able to retain the good elements of previous iterations and use this information to guide the move of the ants in new iteration. The experimental results reveal that the approach can provide better solution quality than the existing metaheuristics in literature.

• Enhance the performance of the evolutionary based metaheuristics by adding an adaptive control mechanism and a guidance mechanism. This contribution is achieved

by developing the Guided Genetic Algorithm (GGA). The adaptive control mechanism is able to effectively adjust the level of intensity and diversity of the population during the optimization process. With the guidance mechanism, which is adapted from our swarm intelligence metaheuristic, the composition of the better individuals is always memorized and updated. Then this information is used to aid the genetic operators to produce offspring solutions with high solution quality. The experimental results showed the benefits of proposed adaptation and guidance mechanisms in enhancing the overall performance of our algorithm.

• Provide an effective and efficient metaheuristic with flexible structure and the ability to maintain the diversity during the exploration of the search space. This contribution is realized by a Multi-agent Self-adaptive Genetic Algorithm (MGA). The algorithm combines the multi-agent system, the hyper-heuristic and the genetic algorithm. The multi-agent systems can provide great improvement to the computational speed and the solution quality. By communicating among different agents, it is able to maintain the diversity of the population after the convergence of the algorithm. The hyper-heuristic is to use to find the most suitable heuristic or sequence of heuristics in a given situation, so the design of each individual heuristic can be flexible. Moreover, with the hyper-heuristic, any new heuristic can be easily inserted into the system without modifying the system's structure, since the hyper-heuristic will select the most appropriate heuristic to apply. For the experiment results, MGA provides the best solutions so far on most of our LTCPP instances.

• Propose a new type of the daily car pooling problem with multiple destinations, provide a new mathematical model and an effective resolution method. Contrary to the classic daily car pooling problem, the servers in multi-destination daily car pooling problem (MDCPP) can pick up clients who go to different destinations, as long as the servers can accept the length of the detour they have to make. Two car pool servers in the model can be given a transfer point, where the clients can change vehicles in order to reach their destinations in time and avoid the server to make long detours. An accurate mathematical model is proposed to this problem, as well as a resolution metaheuristic. The metaheuristic is a hybrid approach based on the ant colony optimization paradigm. A heuristic is designed to locate transfer points between two car pools, and a local search is implemented to further optimize the solution. Experiments are performed to demonstrate the good ability of the approach in solving the MDCPP.

• Design and implement a platform for the test, demonstration, evaluation and comparison of the approaches for solving the LTCPP or other optimization problems (see Appendix 1). The algorithm test and analysis platform provides comparison function to evaluate the results obtained with different parameter settings, so the best one can

be confirmed. It also contains several comparison methods which are designed to analyze the performance of different algorithms. Moreover, a graphical interface is developed to give an intuitionistic view for any manual operation. The solutions generated by the algorithms, as well as the analysis and comparison results are also displayed with graphical visualization, so they can be easily examined. The platform has been proven to effectively facilitate our research process.

• Design and implement a web based platform for the car pooling service of the students of Artois University (see Appendix 1). The platform reveals the value of our research in the real world application. All functions in the long-term car pooling have been achieved in the platform. The participants can register to the platform and submit their requests to find long-term car pool members. The requests then are processed by the algorithms introduced previously and the travel schedules are displayed to the participants with Google API. The platform works well in the real world use. All the requests are successfully solved by the algorithms and remarkable solutions are provided. With the platform, we are able to provide large cost saving for the participants of the car pooling program.

Perspectives

While working on this PhD thesis, some areas to improve further have arisen. They form the basis for future works:

- Some algorithms proposed in this work have some potential for improvement. For instance, the Clustering Ant Colony Algorithm could be enhanced by applying an adaptive evaporate rate similar to the adaptive variation rate of the Guided Genetic Algorithm, in order to provide a better diversity. The Variable Neighborhood Search for the LTCPP could be extended by using a multi-agent system where different agents are assigned with diverse initial solutions, and cooperate by exchanging information related to their searches in the solution space.

- The effectiveness of the developed algorithms on the LTCPP encourages their application to other transportation problems, such as daily car pooling problem, public transportation systems, and scheduling of taxis. Among all the algorithms introduced in this thesis, the multi-agent self-adaptive genetic algorithm is the easiest one to apply to other problems since its structure is the most problem-independent. However, several important aspects may have to be investigated, such as the operator design and the neighborhood definition for each problem.

- The multi-destination car pooling problem can be further studied as well as its solving method. The construction of the transfer points may be embedded into the behavior of the ants instead of using a separate heuristic. In this case, the solution quality of the algorithm could be further improved. However, the computational speed may decrease by this modification, which raises the issue of balancing the solution quality and processing time.

- All our algorithms are designed to provide solutions within a relatively short computing time period, in order to decrease the response time of the car pooling organizer to their users. Recently, the use of graphics processors has been extended to general application domains such as computational science. Indeed, GPUs are very efficient at manipulating computer graphics, and their parallel structure makes them more efficient than general-purpose CPUs for a range of complex algorithms. Thus, modifying the implementation of our algorithm to adapt to the GPU would be a very interesting direction to focus on. This will result in further improvement of the computational speed of our algorithms.

- The two platforms designed in our work still have some improvement space. The

algorithm test and analysis platform are currently able to support other algorithms for solving the long-term car pooling problem, and it can be easily extended to support the algorithms designed for other optimization problems, so the full capacity of the platform can be unleashed. The car pooling platform of Artois University can be performed with further optimization of the interface design and the implementation structure, in order to facilitate the user's operations. Further work may also include the implementation of a mobile phone application which achieves a real time tracking of the participant based on the build-in GPS function of the mobile phones.

The increasing popularity of the long-term car pooling problem and competition among various car pooling service providers require efficient, effective and easy-to-use algorithms for solving the problem, especially for solving the one with large size instances. The approaches proposed in this thesis are very promising and reliable for the real-world implementation. We believe that these algorithms provide valuable suggestions and instructions for the design of the algorithms for different types of the car pooling problem.

APPENDIX 1

Platforms for the Long-term Car Pooling Problem

CONTENTS

Abstract

In this appendix we introduce and demonstrate two platforms: an algorithm test and analysis platform for implementing and evaluating the algorithms we developed for the LTCPP, and a car pooling web platform for servicing the participants of the car pooling program of Artois University. The structure design and implementation detail are presented exclusively. The performance of the two platforms in real-world is concluded.

1.1 Introduction

In this appendix, we present the two platforms we developed during our research: an algorithm test and analysis platform and a car pooling web platform designed for the car pooling program of Artois University in France.

The test and analysis are the key features for the evaluation of an algorithm. During our research, we developed several algorithms for the long-term car pooling problem. For each algorithm, a large amount of experiments and comparisons are required for the parameter setting and the algorithm evaluation, in order to present the most accurate performance of each algorithm. The usual way to implement the algorithms is to implement each of them separately. Therefore, since the algorithms are separately implemented without a management platform, the setting, evaluation and analysis of the parameters of each algorithm and the comparison between the algorithms can only be done manually by several complex, time-consuming and lack of accuracy procedures. Therefore, a standard, open, and scalable test and analysis environment is needed to make the comparison and evaluation work faster, easier, and the most important, more accurate.

Thus, in order to improve our research environment, we propose a light weight and scalable test platform to execute, test, evaluate and compare the algorithms for the long-term car pooling problem. All the algorithms presented in the chapters 2 to 4 are implemented inside the platform, so every algorithm can receive a fair and convincing test environment.

Furthermore, the platform provides comparison function to evaluate the results obtained with multiple combinations of parameters, so the best parameter setting can be confirmed. The platform also contains several comparison methods which provide the analysis to verify the performance of the algorithms. The comparison methods include measurements for typical algorithm evaluation, as well as a few advanced measurements particularly designed for the metaheuristics. Two main comparison modules are developed in the platform. The first comparison module offers the comparison between multiple runs of a single algorithm, the solutions can be compared and analyzed, and the typical statistical values, such as maximum, minimum, average and standard deviation will be calculated and displayed. The second module offers a parallel comparison between the multiple results obtained by different algorithms. Multiple performance aspects of each algorithm are analyzed and compared in detail, and a Friedman test is performed to evaluate the significance of improvement provided by each algorithm. All the information generated by the platform will be recorded in an exclusive report for further use.

Moreover, the analysis and comparison results of the platform are provided with graphical visualization, so they can be easily examined. The graphical interface gives an intuitive view for any manual operation.

The second platform developed in our research is the car pooling web platform which provides car pooling service in real world application. The target user is set to

be the students in the University of Artois.

The deployment of a car pooling service should be supported by multiple features. Given the number and the complexity of the operations to be performed, it is necessary to design a multi-module system as the previous platform, with different modules dedicated to different macro functions, such as user interface module, data management module, optimization module, etc.

Our car pooling platform is designed to support three types of users: car pooling participant, car pooling organizer and system administrator. The car pooling organizer generates and manages the schedules of the car pooling participants. The system administrator guarantees all the features of the platform running properly and provides maintenance service.

All the operations in the car pooling platform are aided with a web based graphical interface. Data collected and used by the car pooling platform are stored in a database with the structure especially designed for the LTCPP. The schedules are generated by an optimization module which contains the Clustering Ant Colony Algorithm and the Multi-agent Self-adaptive Genetic Algorithm introduced in the chapters 2 and 3. However, the car pooling organizer may modify some of the schedules if the participants are not satisfied with them.

The appendix is organized as follows: The algorithm test and analysis platform is introduced in the Section 1.2 in detail manner. Section 1.3 presents our car pooling platform, with the structure, the functions, and the demonstration. In Section 1.4, we conclude with a summary of the main contributions reported in this chapter.

1.2 Algorithm Test and Analysis Platform

A user case diagram is given in Figure 1.1 to show the main functions of this platform. The architecture of the algorithm test and analysis platform is consisting of four modules: an algorithm box module, an algorithm analysis module, a database module and a graphical user interface.

The algorithm box module is designed to implement the algorithms designed for the long-term car pooling problem. The module contains the four algorithms presented in the chapters 2 to 4. Also, two other algorithms designed by other authors are also implemented in the platform in order to be compared with our approaches in a fair environment. The algorithms are listed in Figure 1.2. User can switch among all six algorithms in order to test or solve a long-term car pooling problem. The module is designed with good expandability, thus it is very easy to insert any new algorithm into this module or remove any existing algorithms from the module.

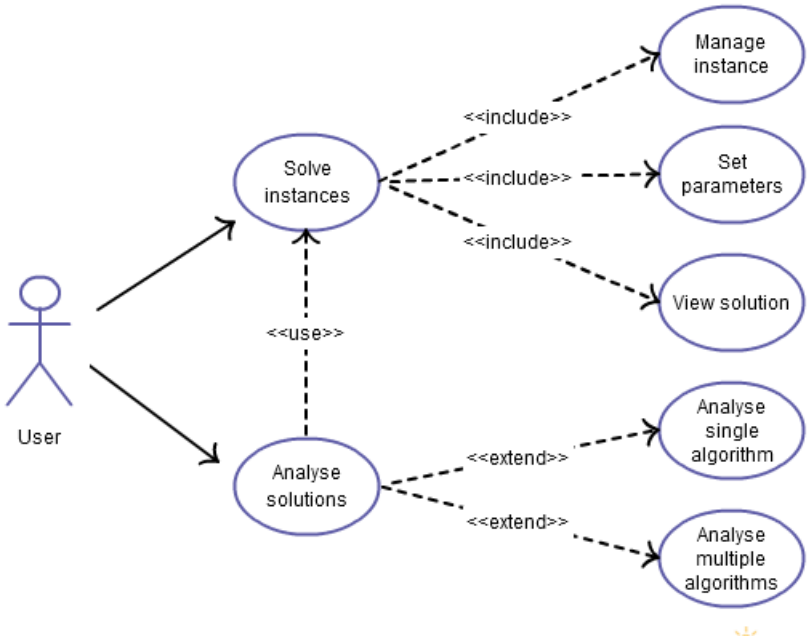


Figure 1.1: The user case diagram of the algorithm test and analysis platform

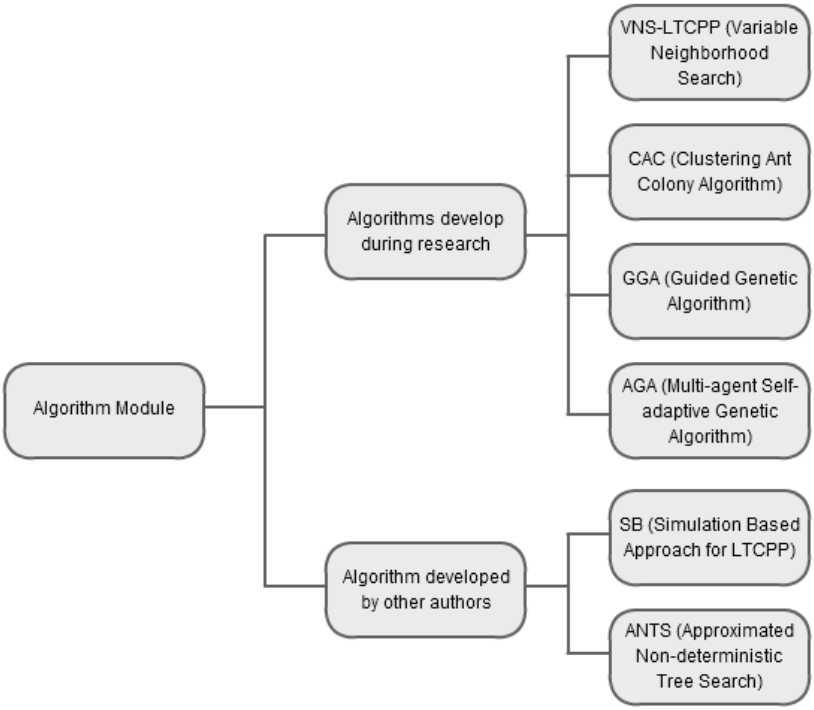


Figure 1.2: Algorithms implemented in the algorithm box module

The algorithm analysis module provides the functions for the comparison and evaluation of the algorithms. This module consists of two main sub-modules. The first sub-module analyzes the results obtained by a single algorithm. Both experiments with single run and multiple runs can be analyzed. This sub-module is mainly designed to aid the parameter setting of an algorithm. The second sub-module covers the

analysis of the results obtained by multiple algorithms. It can generate multiple performance comparisons among the algorithms in the algorithm box module. This sub-module dedicates to the performance evaluation of different algorithms.

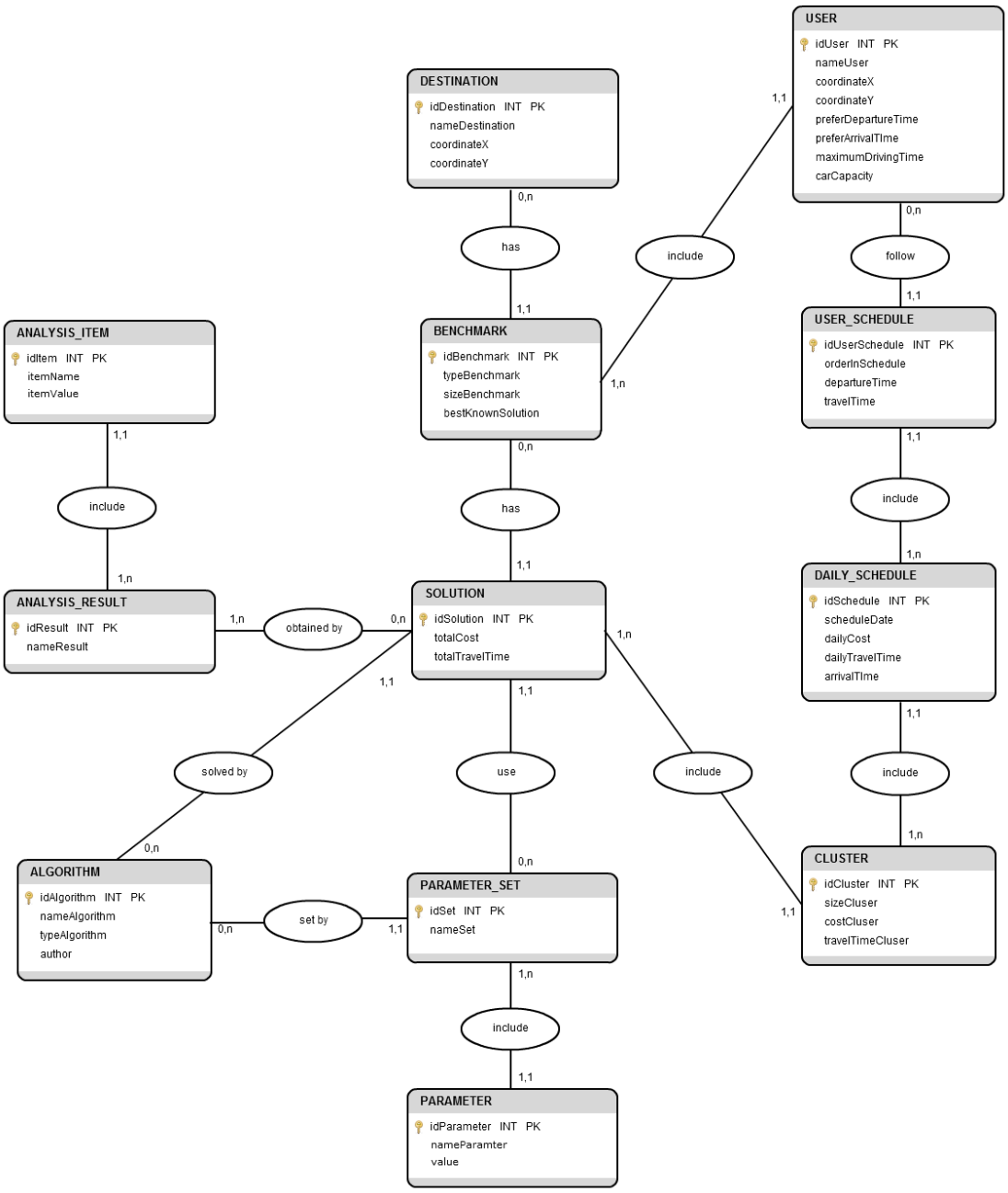


Figure 1.3: The structure of the database design

The database module is implemented in MySQL by means of a standard relational DBMS structure. Relational data contains all elements relevant to the platform, including the management of the benchmarks, the parameter settings of all the algorithms and the analysis results of the algorithm analysis module. The design of the database structure is shown in Figure 1.3.

The Graphical User Interface (GUI) is implemented to provide the functions of the data input and result display. The data input includes the benchmark selection, the parameter setting of each algorithm and the result analysis, while the result display demonstrates the solution obtained by the algorithm, including the cluster distribution, the user schedules and the statistic information, and the analysis results, such as the solution curve, bar chart and analysis result tables. The module allows accessing the database to read and write the experimental results and benchmarks.

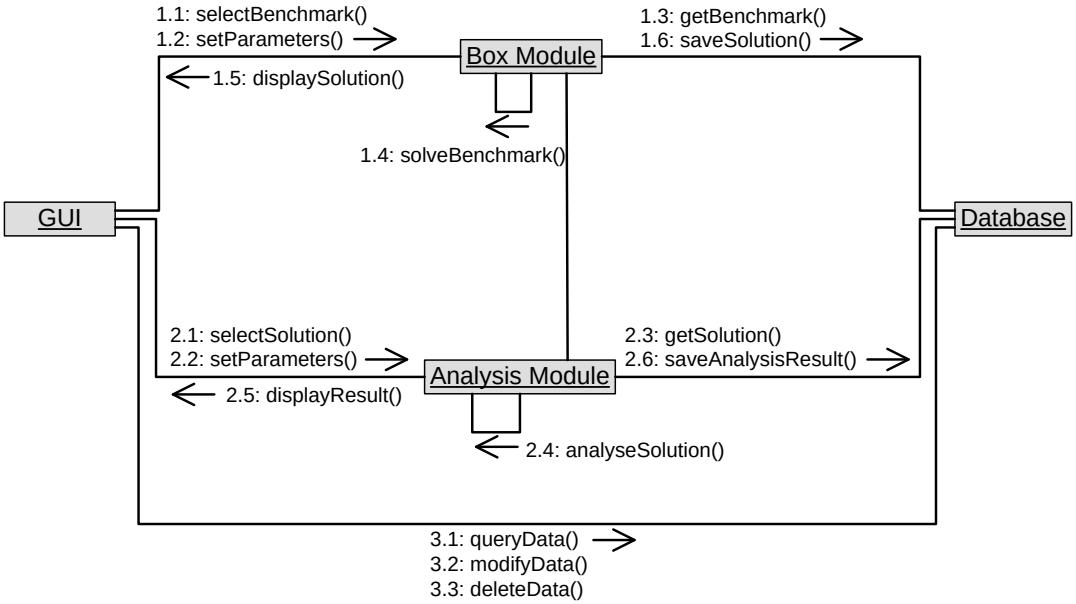


Figure 1.4: The collaboration diagram of the algorithm test and analysis platform

The interactions between the GUI and other components of the platform are presented in Figure 1.4. The user selects benchmarks and sets parameters for the algorithms in the algorithm box module, and then the solutions are returned to GUI and displayed to the user after the benchmarks are solved. The solutions, at the same time, are also saved to the database module. With the parameters set through the GUI, the analysis module performs the evaluation of the solutions and displays the analysis results in the GUI. All the data, such as the benchmarks, the optimization results and the analysis results are stored in the database module which is accessible through the GUI, so the user can easily modify and remove any data in the database.

1.2.1 Function Workflow

As abovementioned, the two main functions of the platform are to solve LTCPP instances and to analyze the results. Thus, in this section, we present the detail workflow analysis of these two functions.

A. Solving a LTCPP instance

The platform's workflow to solve a LTCPP instance is presented in Figure 1.5. The first step of the operations is to select a benchmark for the platform. The selected benchmark will be verified by the system. If the benchmark is a legal LTCPP instance, the process will continue, otherwise the system requires selecting another benchmark.

After the benchmark is confirmed, the user has to select an algorithm to solve the problem from the candidate algorithms, such as VNS, CAC, GGA... Afterward, the user can either set the parameters of the algorithm himself or use the best-known parameter setting stored in the system. For the former case, the parameters will be examined by the system, and if the parameters are not feasible, the system will require the user to modify the setting referred to the suggestion given by the system. When the parameter setting is considered legal, the system will start to solve the problem, and then display the experimental results.

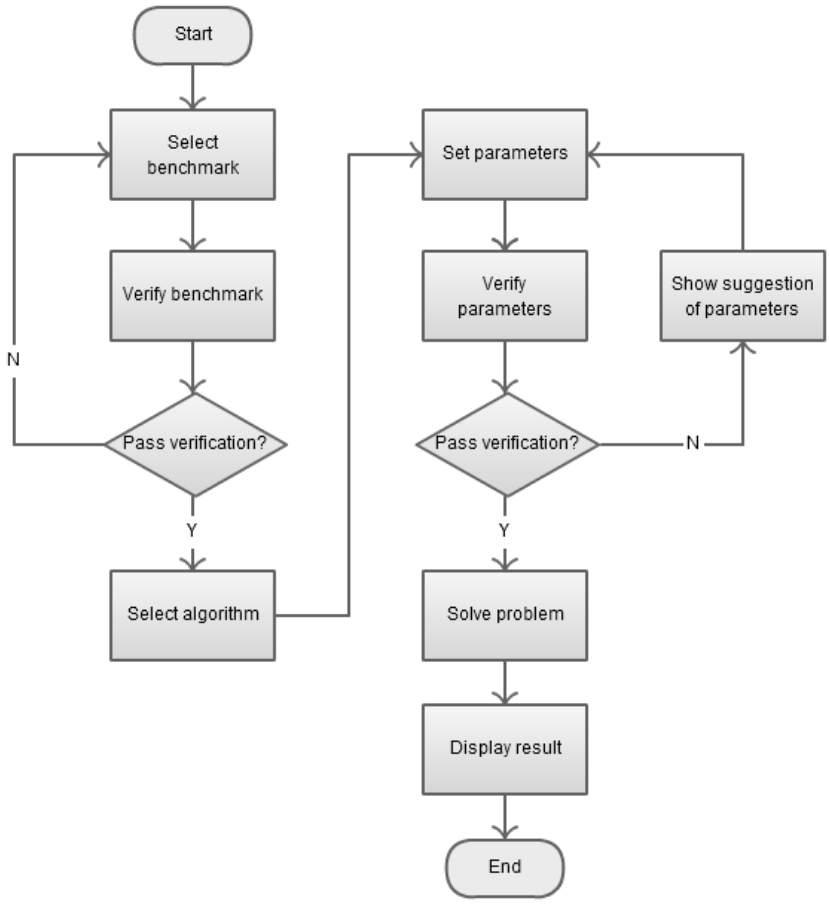


Figure 1.5: The workflow of the function of solving a LTCPP instance

B. Performing a result analysis

To start the result analysis, the user first need to select the analysis type, the analysis can be based on a single run of a single algorithm, or multiple runs of a single algorithm, or multiple results of multiple algorithms. For each type of analysis, user then has to specify a set of parameters, such as the methods of comparison, the aspects of comparison and the type of chart to display the analysis results. Then the analysis will be performed. Afterward, the analysis results will be displayed according to the requirement of the user, and functions of saving the result and export the result as a report are also available to the user. The workflow of the platform to perform a result analysis is shown in Figure 1.6.

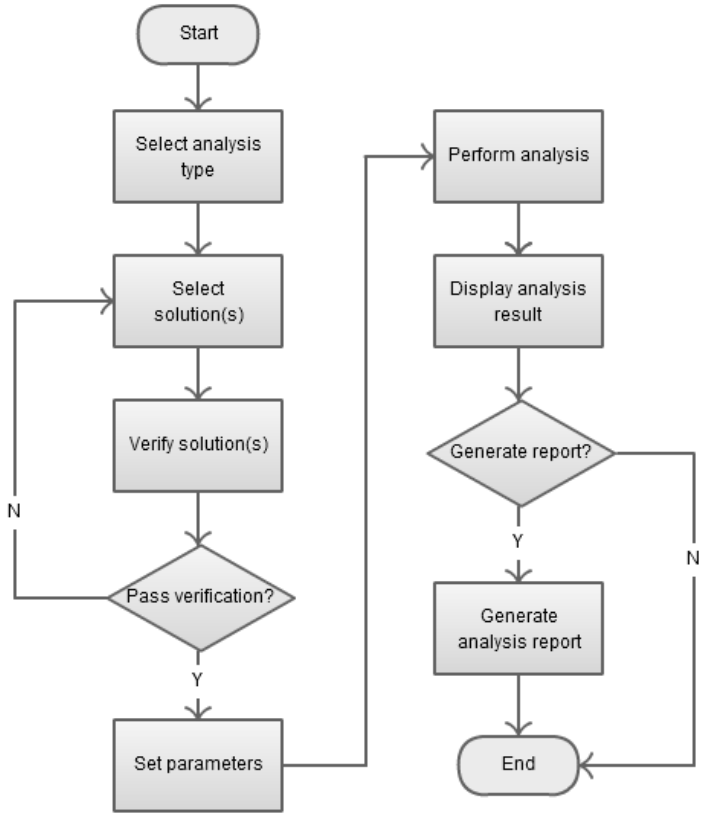


Figure 1.6: The workflow of the function of result analysis

1.2.2 Demonstration of the platform

The algorithm test and analysis platform is demonstrated in this section. The demonstration presents some important interfaces which users may meet frequently during their operations. This section is presented in four parts: benchmark selection, algorithm parameter setting, optimization result display and analysis result display. Figure 1.7 shows the interface for benchmarks selection and algorithm parameter setting.

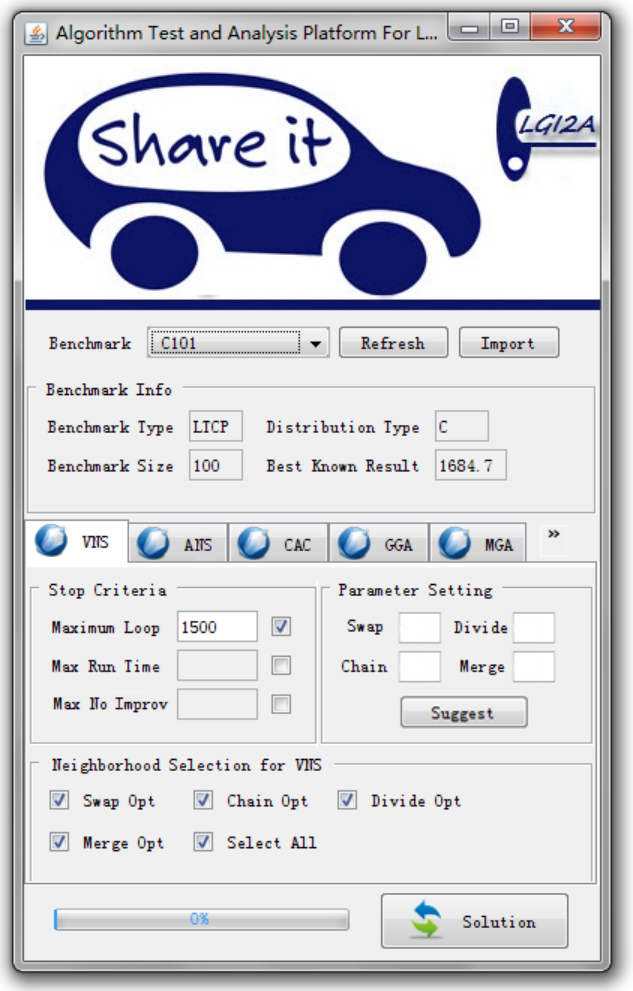


Figure 1.7: The interface for benchmarks selection and algorithm parameter setting

A. Benchmark Selection

The benchmark selection includes the following functions:

- Select: This function provides benchmarks for users to select. All the available benchmarks are obtained from the database and put into a combo list.
- Refresh: the function is used when the benchmarks in the database are modified. The function updates the modifications in the database.
- Import: the function is designed to import new benchmarks into the database. Currently only the database format benchmark can be imported.
- Benchmark info: This area shows the detail information of the selected benchmark, such as the benchmark type, the distribution of the users in the benchmark, the amount of users of the benchmark and the value of the best known solution of the benchmark.

B. Algorithm Parameter Setting

Six algorithms are implemented in the platform, the user can switch among them to solve the long-term car pooling problem. The algorithm is currently displayed on the top layer is considered selected. For each algorithm, we design an algorithm dependent panel for users to input the necessary parameters. The items in the panels are introduced as follows.

- Stop criteria: the system provides three criteria to terminate an algorithm: maximum iterations, maximum run time, and no improvement during a user-defined number of iterations. User can select any of them by ticking the box next to each option.
- Parameter setting: user can set all necessary parameters of the algorithm in this area. A suggestion button is provided. The parameters will be filled with the best known parameter setting by clicking it.
- Neighborhood selection: most of the implemented approaches are embedded with the variable neighborhood search (VNS) procedure. This area enables the user to turn on and off an individual operator in the VNS. Thus, different combinations of operators can be achieved.

C. Optimization Result Display

This part demonstrates the solution of a benchmark. The general resolution results are displayed for each run, including the total cost, computing time, amount of iterations, amount of the car pools built and amount of traveled alone users. Moreover, the platform also provides a graphical view of the distribution of the clusters and the detail time schedule of each car pool.

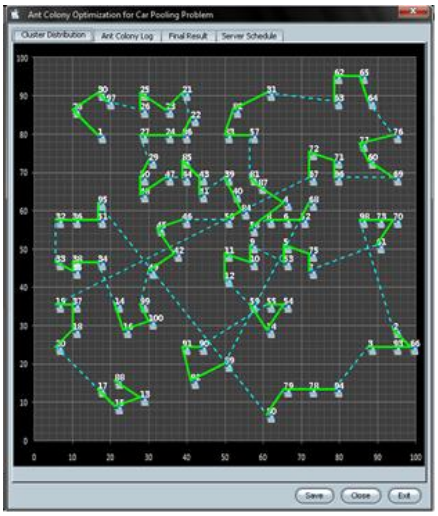


Figure 1.8 (a): Cluster distribution of the clustering ant colony algorithm

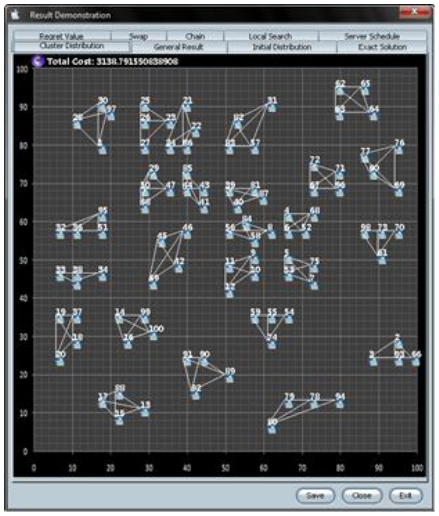


Figure 1.8 (b): Cluster distribution of the guided genetic algorithm

The distribution of the clusters is displayed based on the algorithm used to solve a problem. Figure 1.8 (a) presents the cluster distribution of a solution obtained by the clustering ant colony algorithm. The graph shows the construction trajectory of the solution, where the moves between the users are marked in green, and the moves between the clusters are indicated by blue. By viewing the graph, the steps of the solution construction can be examined easily. Figure 1.8 (b) shows the cluster distribution of a solution obtained by the guided genetic algorithm. Since the clusters in GGA are obtained by modifying the chromosome instead of constructing a solution path, the graph therefore only displays the composition of each cluster and its inner paths.

The user schedule is also presented by the platform. The display consists in the schedule when each user acts as a server, which includes the order and time for the current server to pick up other car pool members. The pickup time will pop up when the cursor points at the corresponding user. Figure 1.9 shows an example of the user schedule display, note that in each cluster, every user has to act as a server.

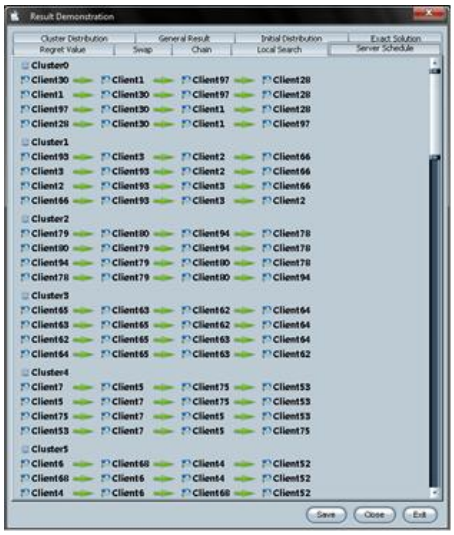


Figure 1.9: The user schedule display

D. Analysis Result Display

This part demonstrates three types of solution analysis provided by the platform.

Figure 1.10 shows the analysis of a single run of an algorithm. The analysis reveals the optimization curve, the solution quality, the computing time, the iteration number where each global best solution is improved, and the comparison to the best-known solution of the instance.

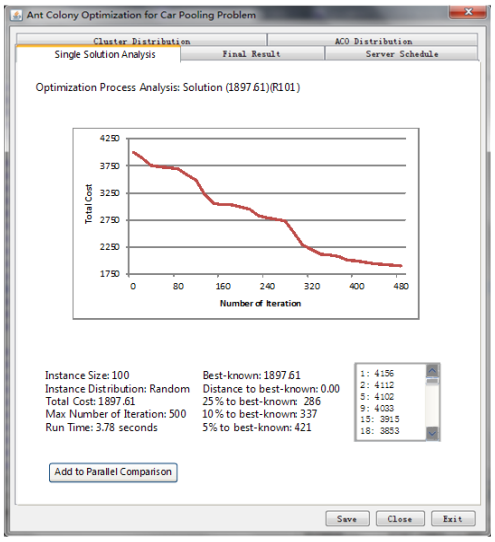


Figure 1.10: Single run analysis of an algorithm

Figure 1.12 presents the analysis of multiple runs of an algorithm. The example reveals the comparison between the solution qualities of different runs. A bar chart is used to display the comparison result. Moreover, the maximum, minimum, average, standard deviation values of the runs, and the distance between the average and the best known solution are given below the chart. This analysis is mainly used for deciding the parameter setting of an algorithm and for performing an initial review of an algorithm for the multiple algorithm comparison analysis.

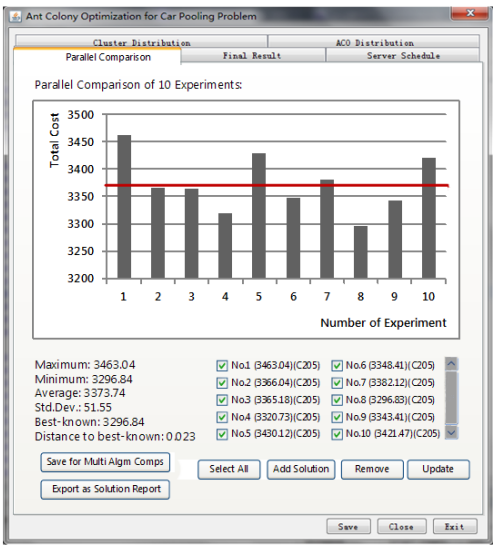


Figure 1.11: Multiple runs analysis of an algorithm

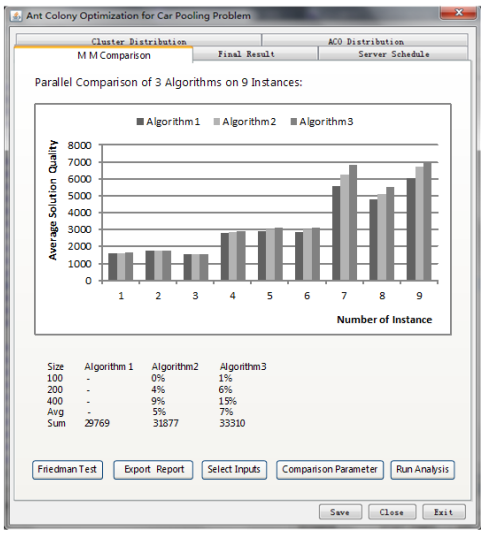


Figure 1.12: Multiple algorithms analysis

Figure 1.12 displays the analysis of multiple runs of multiple algorithms. The solution costs obtained by different algorithms are compared and shown in bar chart. The data for the comparison can be imported from the single algorithm multiple runs analysis. The algorithms are ordered and ranked according to their solution quality. The solution quality differences between the algorithm with the best performance and other algorithms are calculated and displayed in a table. By clicking the 'Friedman test' button on bottom left, an exclusive comparison report performed with the Friedman test method can be generated.

1.2.3 System requirements

The software and hardware requirements for running the algorithm test and analysis platform are as follows.

- Windows XP or higher operating system / Linux 5.0 or higher operating system
- Java environment 1.4 or higher
- 1GHz CPU or higher
- 512M RAM or higher
- Minimum screen Resolution of 800x600 pixels and a minimum color depth of at 256 colors.

1.2.4 Summary

This section presents our test and analysis platform. The platform is designed to

implement and analyze the algorithms we designed for solving the long-term car pooling problem. The platform provides a fair and convenient environment to perform efficiently evaluation and comparison of the algorithms. The graphical user interface successfully displays the detailed solutions of LTCPP instances and the analysis results of algorithms. The platform has been proven to be a very useful tool for our research of the long-term car pooling problem.

Furthermore, to the best of our knowledge, our test and analysis platform is the first attempt to implement such a platform for the long-term car pooling problem. In future works, our test and analysis platform will be extended to support other algorithms for other optimization problems.

1.3 Car pooling platform

The car pooling platform is designed for an actual use in real-world for students in the Artois University. The platform is a web application developed with PHP and HTML language and connected to the Google Map Web service. Two of the algorithms described in the previous chapters are embedded in the platform to provide solutions for the users. A user case diagram of the platform is presented in Figure 1.13.

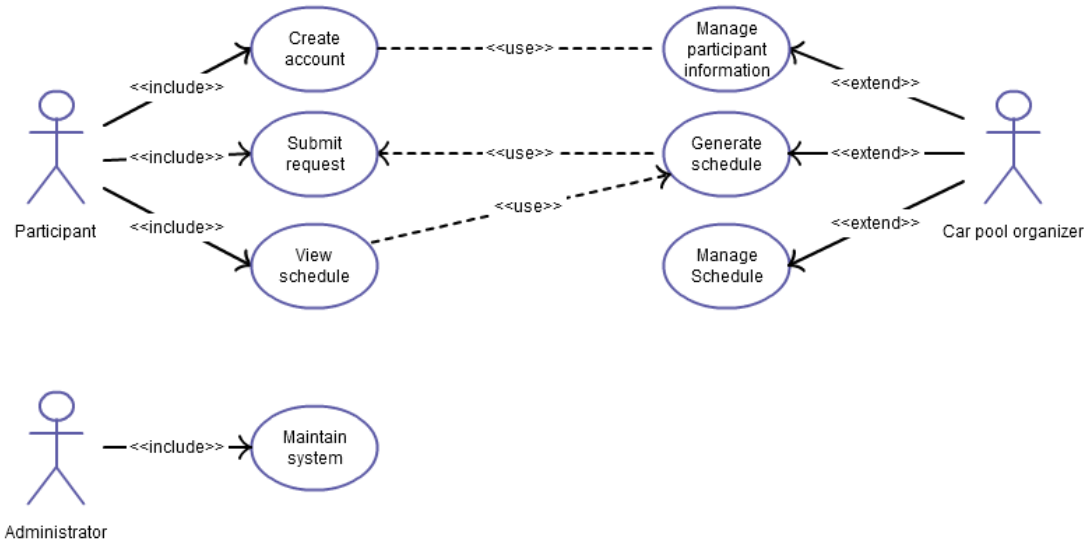


Figure 1.13: The user case diagram of the car pooling platform

The platform consists in a solution building module, a database module and a webpage based graphical user interface. The three modules are implemented as follows.

The solution building module contains two algorithms: the clustering ant colony algorithm and the multi-agent self-adaptive genetic algorithm. The former algorithm is preferable in solving the instances with less than 200 users, whereas the latter algorithm is favored to solve the cases with more than 200 users.

The database module is designed to store all related data of the platform, including the geographical information of users, the requests of users, the solution information and the detail travel schedule of users. The design of the database is shown in Figure 1.14.

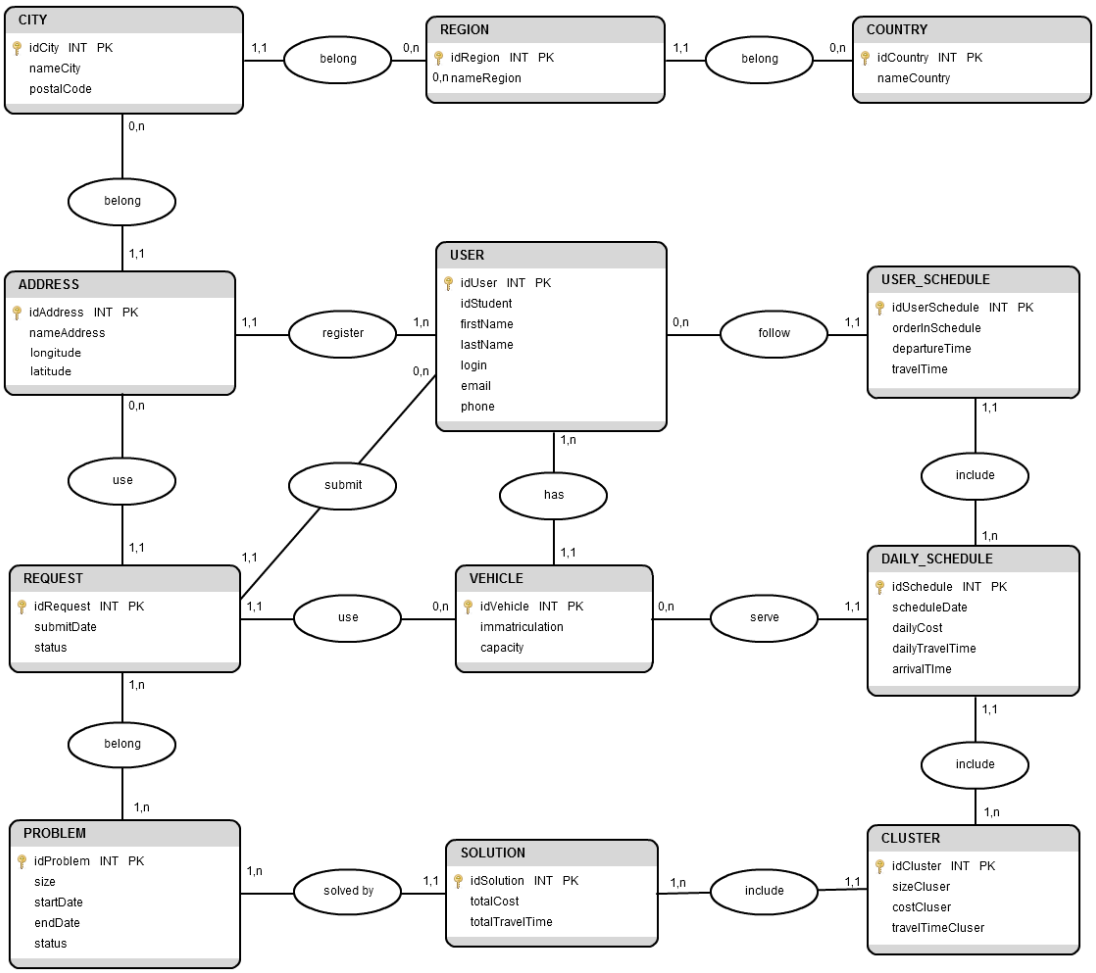


Figure 1.14: Database structure of the car pooling platform

The graphical user interface is implemented by using web application. It permits the users to input all information related to the participation of the car pooling program, to access the database and to display the car pooling schedule in Google Map. According to their distinct requirements of functions, different interfaces are designed for the car pooling participant and the car pooling organizer. The interactions between the interfaces and the other two modules are presented in Figure 1.15. The main functions corresponding to a car pooling participant are account creation, car pooling request submission and detailed car pooling schedule display. The functions concerning the car pooling organizer are the management of the participants' information and their schedules.

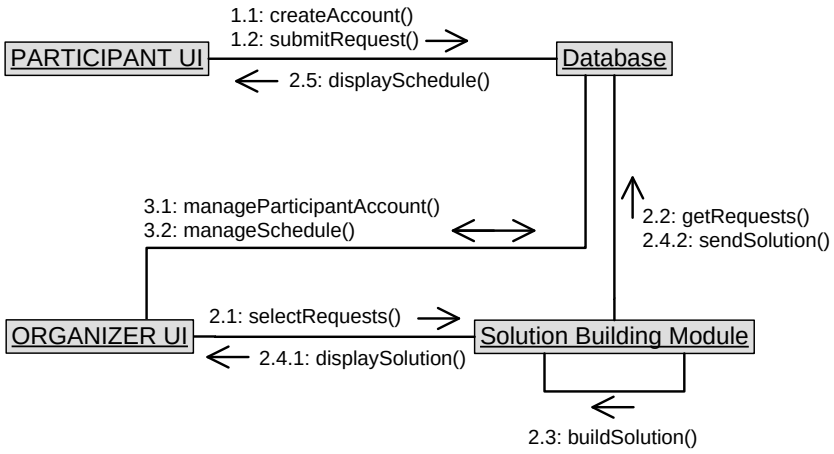


Figure 1.15: Interaction diagram of the components of the car pooling platform

1.3.1 Development techniques

Google Maps API is essential in our platform development. It is designed to allow developers to integrate Google Maps into their websites. By using the Google Maps API, it is possible to embed Google Maps site into an external website, on to which site specific data can be overlaid. The Maps API includes an API for Adobe Flash applications, a service for retrieving static map images, and web services for performing geocoding, generating distances, and obtaining elevation profiles. The Google Maps API is free for commercial use providing that the site on which it is being used is publicly accessible and does not charge for access.

1.3.2 Demonstration of the platform

The main functions of the car pooling platform are demonstrated in this section.

A. Create an account

In order to generate high quality schedules, the car pooling platform requires some coordination and personal information from participants. Thus, each user of the platform is required to create a functioning account with accurate information. The inputs of the user to create the account include: login, password, name, gender, student card number, email, cellphone number, a valid home address, driver license, and vehicle capacity.

Furthermore, all users of the platform must respect certain ethical and safety rules. This laid out the responsible behavior that users must adopt, which entails respecting

the rules of good conduct, and of safety with regard to other users. Figure 1.16 shows the interface to aid the user to input a valid address.

Figure 1.16: the interface to aid the user to input a valid address.

B. Submit a request

To submit a request to the platform, a user first has to confirm the departure location and destination, and then inputs the date when the user will participate the car pooling program, the earliest departure time, latest the arrival time, and the maximum travel time. Note that a user can submit multiple requests as long as the periods of each are not over lapped. Figure 1.17 shows the interface to input the acceptable departure and arrival time ranges.

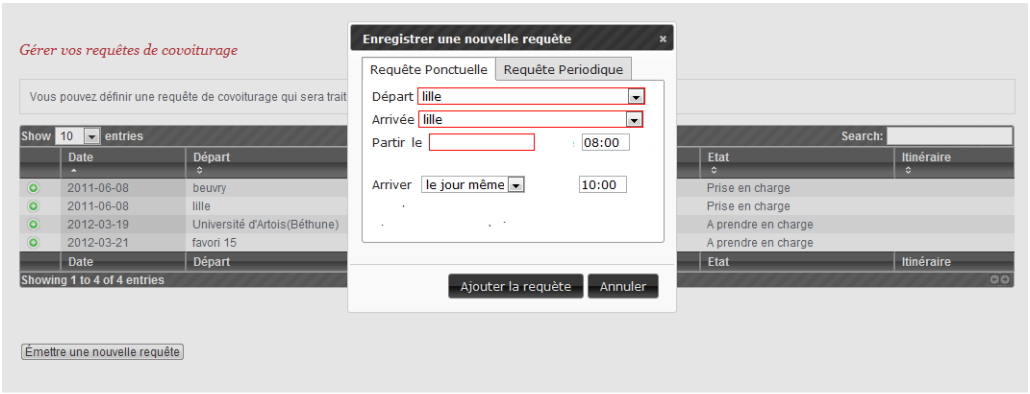


Figure 1.17: the interface to input the departure and arrival time.

C. Receive a schedule

After the requests are solved by the resolution algorithm, the schedule of each particular participant will be sent. Since the platform is for long-term car pooling, the partici-

pant is able to view all the daily schedules of his participation period. In each daily schedule, the user can check the pickup time and order of each car pool member. The total driving time and distance are also displayed along with route presented in the Google Map. The user may also view the percentage of cost being saved by participating the car pool program as well as the extra travel time the user has to spend. Figure 1.18 presents the interface showing the detail schedule of a car pooling participant.

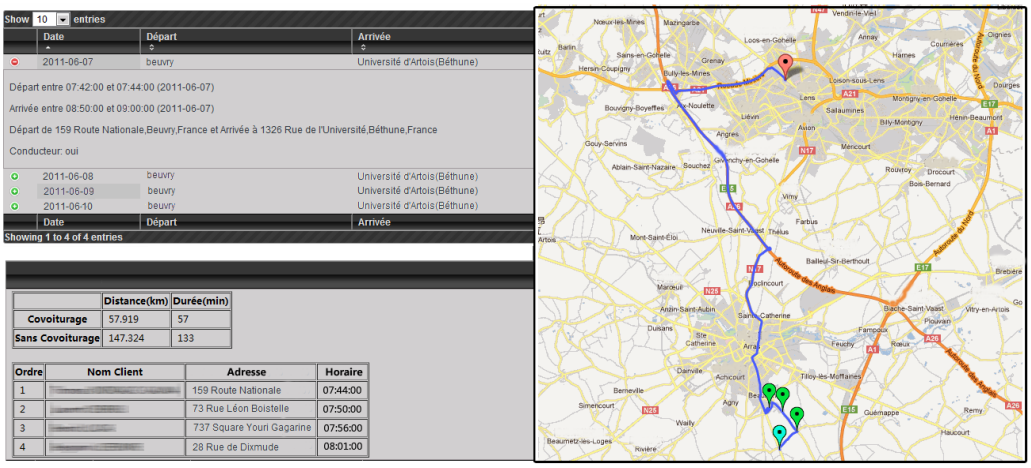


Figure 1.18: the interface showing the detail schedule of a car pooling participant.

D. Manage the participants and schedules

This function is only available for the car pooling organizers. They are authorized to modify and delete the participants' information and the car pooling schedules. Figure 1.19 and 1.20 shows the organizer view of the participants' address information and the car pooling schedules

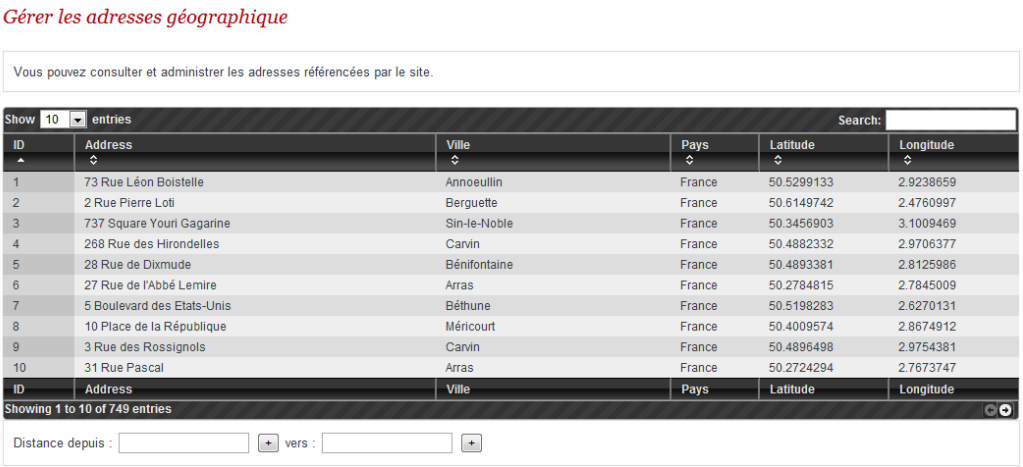


Figure 1.19: the organizer view of the participants' address information.

Figure 1.20: the organizer view of the car pool schedules.

1.3.3 System requirements

The software and hardware requirements for running the car pooling platform are specified as follows.

- Windows XP or higher operating system / Linux 5.0 or higher operating system
- 1GHz CPU or higher
- 512M RAM or higher
- Minimum screen Resolution of 800x600 pixels and a minimum color depth of at 256 colors.
- Internet explorer 6.0 or higher
- Java environment 1.4 or higher
- MySQL 4.0 or higher
- Apache 2.0 or higher

1.3.4 Summary

In this section, the car pooling platform in presented. The platform is designed and implemented to serve the students in our university, thus it is a real world application. All functions in the long-term car pooling have been achieved in the platform. The participants can register in the platform and submit their requests to find long-term car pool members. The requests then are processed by the algorithms introduced previously and the travel schedules are displayed to the participants.

The platform works well in the real world use. We obtained more than 500 registered users and more than 700 requests. The requests are successfully solved by the algorithms and remarkable solutions are provided. According to our statistic result, we are able to provide an average of 67% cost saving for the participants with only an average of 16% extra travel time. Since the students in the same university have the same time to start their first course, which results in similar arrival time window, building a car pool becomes relatively easy comparing with the car pool program with random participants. Our car pool platform has effectively provided car pool matching for 88% of all the participants. Only 12% of the registered users are not able to find

car pool members.

The implementation of the car pooling platform is considered a success. Further improvements of the platform consists in the implementation of a mobile phone application which can show all the car pool information for the user and track user's real time location with the aid of the GPS function.

1.4 Conclusion

The two platforms we developed during our research are introduced in this appendix. The algorithm test and analysis platform facilitates our research process by providing an exclusive result display and analysis function, while the car pooling platform reveals the value of our research in the real world application. Both platforms are proven to work effectively.

The algorithm test and analysis platform are currently able to support other algorithms for solving the long-term car pooling problem, and the future work is to extend the platform to support the algorithms designed for other optimization problems, in order to unleash the full capacity of the platform.

The car pooling platform will be performed with further optimization in the implementation, in order to gain faster response speed to user's operations. Further work also includes the implementation of a mobile phone application which achieves a real time tracking of the participant based on the build-in GPS of the mobile phone.

APPENDIX 2

Using Friedman Test to Compare the Performance of Metaheuristics

CONTENTS

2.1 Procedure of Friedman test

The Friedman test can be used to compare the performance of k ($k \geq 2$) metaheuristics using a test bed with b instances [Conover, 1998][Villegas, 2011]. The data of the experiment will be presented in a $b \times k$ table, where each entry represents the result of the objective function found by the metaheuristics, as shown in table 2.1.

The procedure to perform the Friedman test is as follows:

- Rank the results of the metaheuristics within each instance, giving 1 to the best and k to the worst. Let $R(X_{ij})$ be the rank, from 1 to k, assigned to X_{ij} of instance i.
- Calculate the total summation of squared ranks A_2 with equation (2.1).

$$A_2 = \sum_{i=1}^{b} \sum_{j=1}^{k} [R(X_{ij})]^2 \tag{2.1}$$

Instance	Metaheuristic			
	Metaheuristic 1	Metaheuristic 2	…	Metaheuristic k
Instance 1	X_{11}	X_{12}	…	X_{1k}
Instance 2	X_{21}	X_{22}	…	X_{2k}
…	…	…	…	…
Instance b	X_{b1}	X_{b2}	…	X_{bk}

Table 2.1: Presentation of data for the Friedman test

In the absence of ties A_2 simplifies to equation (2.2).

$$A_2 = \frac{bk(k+1)(2k+1)}{6} \tag{2.2}$$

We calculate the summation of the rank for each metaheuristic using equation (2.3) and then calculate B_2 with equation (2.4).

$$R_j = \sum_{i=1}^{b} R(X_{ij}) \tag{2.3}$$

$$B_2 = \frac{1}{b}\sum_{j=1}^{k} R_j^2 \tag{2.4}$$

Then, the test statistic is given by equation (2.5).

$$T_2 = \frac{(b-1)(B_2 - bk(k+1)^2/4)}{A_2 - B_2} \tag{2.5}$$

Using a F distribution table, we can find the critical value $F_{1-\alpha,k-1,(b-1)(k-1)}$ with a significance level α. If T_2 is greater than $F_{1-\alpha,k-1,(b-1)(k-1)}$, we reject the null hypothesis, and then there exist at least one metaheuristic whose performance is different from at least one of the other metaheuristics.

However, it is necessary to perform paired comparisons to know which metaheuristics are really different if there are more than two metaheuristics being compared.

2.2 Procedure of paired Comparisons

The paired comparison is used to know if metaheuristics *i* and *j* are considered different after the rejection of the null hypothesis with the Friedman test.

We calculate the absolute difference of the summation of the ranks of metaheuristics *i* and *j*, and if constraint (2.6) is satisfied, we declare that metaheuristics *i* and *j* are different.

$$\left|R_i - R_j\right| > t_{1-\frac{\alpha}{2}}\left[\frac{2b(A_2 - B_2)}{(b-1)(k-1)}\right]^{\frac{1}{2}} \tag{2.6}$$

where $t_{1-\alpha/2}$ is the *1-α/2* quantile of the *t* distribution with *(b-1)(k-1)* degree of Freedom.

2.3 Comparisons of the approaches in this thesis

In this section, we demonstrate in detail the comparisons of the approaches we presented in this thesis. The demonstration is organized according to the chapters.

2.3.1 Comparison in Chapter 2

In chapter 2, we compare our VNS-LTCPP approach with the SB approach. The ranks of the solution quality of each approach for each instance are presented in table 2.2.

Instance	VNS-LTCPP			SB [Correia, 2007]		
	Avg	R	R^2	Avg	R	R^2
C101	1684.6	2	4	1669.2	1	1
C102	1753.8	2	4	1724.8	1	1
C103	1563.6	1	1	1599.4	2	4
C201	2723.8	1	1	2868.6	2	4
C202	3145.2	2	4	3114.1	1	1
C203	2993.7	1	1	3182.4	2	4
C401	6130.1	1	1	6860.3	2	4
C	5110.7	1	1	5524.5	2	4
C403	6322.5	1	1	6994.5	2	4
R101	2286.6	2	4	2265.4	1	1
R102	1898.7	1	1	2091.7	2	4
R103	2379.8	1	1	2418.5	2	4
R201	4464.6	1	1	4567.1	2	4
R202	4162	1	1	4283.3	2	4
R203	4282.5	2	4	4257.5	1	1
R401	8398.4	1	1	8993.8	2	4
R402	6975	1	1	7417.5	2	4
R403	8422	1	1	8933.5	2	4
W101	886.9	2	4	885.7	1	1
W102	1041.5	2	4	1037.9	1	1
W103	1173.8	1	1	1187.6	2	4
W201	1682.5	1	1	1722.9	2	4
W202	2003.6	1	1	2127.7	2	4
W203	1806.8	1	1	1965.1	2	4
W401	3076.8	1	1	3442.8	2	4
W402	3513	1	1	3984.9	2	4
W501	5288.4	1	1	5858.3	2	4
Avg		1.25			1.74	
Sum		34	48		47	87

Table 2.2: Ranks between the VNS-LTCPP and the SB approach.

In this comparison, the number of approaches $k = 2$, and the number of instances $b = 27$. Thus, $A_2 = 48 + 87 = 135$, while $B_2 = (34^2+47^2)/27 = 124.63$.

So the T_2 value is calculated as follows:

$$T_2 = \frac{(27-1)(124.63 - 27\times 2\times 3^2/4)}{135-124.63} = 7.85$$

From the F distribution table, with 1-0.01 quantile, $F_{1-\alpha,k-1,(b-1)(k-1)} = F_{0.99,1,26} = 7.72$. Since $T_2 > F_{0.99,1,26}$, we reject the null hypothesis, then there exist at least one approach whose performance is different from at least one of the other metaheuristics. Because there are only two approaches being compared, and the summation of the ranks of the VNS-LTCPP approach is smaller than the one of the SB approach, it is clear that the VNS-LTCPP approach is significant better than the SB approach.

2.3.2 Comparison in Chapter 3

In chapter 3, we compare our CAC algorithm with the ANTS algorithm, the SB approach and the VNS-LTCPP presented in chapter 2. The ranks of the solution quality of each approach for each instance are presented in table 2.3.

In this comparison, the number of approaches $k = 4$, and the number of instances $b = 27$. Thus, $A_2 = 55+173+344+238 = 810$, while $B_2 = (35^2+65^2+92^2+78^2)/27 = 740.67$.

So the T_2 value is calculated as follows:

$$T_2 = \frac{(27-1)(740.67 - 27\times 4\times 5^2/4)}{810-740.67} = 24.63$$

From the F distribution table, with 1-0.01 quantile, $F_{1-\alpha,k-1,(b-1)(k-1)} = F_{0.99,3,78} = 4.04$. Since $T_2 > F_{0.99,3,78}$, we reject the null hypothesis, then there exist at least one approach whose performance is different from at least one of the other metaheuristics.

A paired comparison is then performed to decide which approaches are really different. From a T distribution table, $t_{1-\alpha/2}$ for $\alpha=0.01$ and (b-1)(k-1) = 78 degrees of freedom is 2.64. Thus, the critical value for the difference is:

$$t_{1-\frac{\alpha}{2}}\left[\frac{2b(A_2-B_2)}{(b-1)(k-1)}\right]^{\frac{1}{2}} = 2.64\times\left[\frac{2\times 27\times(810-740.67)}{26\times 3}\right]^{\frac{1}{2}} = 18.29$$

Table 2.4 shows the absolute difference of the summation of the ranks between every two approaches. We can see that our CAC approach provides the best performance. It outperforms significantly all the other approaches.

Instance	CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]			VNS-LTCPP		
	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2
C101	1593.4	2	4	1592.9	1	1	1669.2	3	9	1684.6	4	16
C102	1728.2	2	4	1748.5	3	9	1724.8	1	1	1753.8	4	16
C103	1527.5	1	1	1535.1	2	4	1599.4	4	16	1563.6	3	9
C	2717.7	1	1	2854.2	3	9	2868.6	4	16	2723.8	2	4
C202	2892.9	1	1	3004.5	2	4	3114.1	3	9	3145.2	4	16
C203	2834.1	1	1	3003.5	3	9	3182.4	4	16	2993.7	2	4
C401	5618.6	1	1	6281.4	3	9	6860.3	4	16	6130.1	2	4
C402	4760.3	1	1	5153.2	3	9	5524.5	4	16	5110.7	2	4
C403	6046.4	1	1	6742.1	3	9	6994.5	4	16	6322.5	2	4
R101	2283.2	3	9	2281.5	2	4	2265.4	1	1	2286.6	4	16
R102	1874.3	2	4	1864.2	1	1	2091.7	4	16	1898.7	3	9
R103	2313.4	1	1	2438.7	4	16	2418.5	3	9	2379.8	2	4
R201	4231.8	1	1	4253.5	2	4	4567.1	4	16	4464.6	3	9
R202	3824.2	1	1	4071.9	2	4	4283.3	4	16	4162	3	9
R203	4304.6	3	9	4541.5	4	16	4257.5	1	1	4282.5	2	4
R401	8033.7	1	1	8580.4	3	9	8993.8	4	16	8398.4	2	4
R402	6559.7	1	1	6893.3	2	4	7417.5	4	16	6975	3	9
R403	8129.5	1	1	8338.9	2	4	8933.5	4	16	8422	3	9
W101	886.3	2	4	893.8	4	16	885.7	1	1	886.9	3	9
W102	1008.5	1	1	1020.3	2	4	1037.9	3	9	1041.5	4	16
W103	1134.8	1	1	1168.1	2	4	1187.6	4	16	1173.8	3	9
W201	1557.6	1	1	1601.8	2	4	1722.9	4	16	1682.5	3	9
W202	1812.9	1	1	1919.2	2	4	2127.7	4	16	2003.6	3	9
W203	1784.4	1	1	1907.5	4	16	1965.1	3	9	1806.8	2	4
W401	2848.4	1	1	3066.4	2	4	3442.8	4	16	3076.8	3	9
W402	3360.1	1	1	3692.9	3	9	3984.9	4	16	3713	2	4
W501	5056.2	1	1	5224.9	2	4	5858.3	4	16	5288.4	3	9
Avg		1.30			2.41			3.41			2.89	
Sum		35	55		65	173		92	344		78	238

Table 2.3: Ranks among the CAC, the ANTS, the SB and the VNS-LTCPP.

$\lvert R_i-R_j \rvert$	ANTS [Maniezzo, 2004]	SB [Correia, 2007]	VNS-LTCPP
CAC	30	57	43
ANTS [Maniezzo, 2004]	-	27	13
SB [Correia, 2007]	-	-	14

Table 2.4: Paired comparison results.

2.3.3 Comparison in Chapter 4

In chapter 4, we first compare our GGA algorithm with the CAC approach presented in chapter 3, the ANTS algorithm and the SB approach. The ranks of the solution quality of each approach for each instance are presented in table 2.5.

Instance	GGA			CAC			ANTS [Maniezzo, 2004]			SB [Correia, 2007]		
	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2
C101	1599.3	3	9	1593.4	2	4	1592.9	1	1	1669.2	4	16
C102	1712	1	1	1728.2	2	4	1748.5	3	9	1724.8	4	16
C103	1543.9	3	9	1527.5	1	1	1535.1	2	4	1599.4	4	16
C201	2749.4	2	4	2717.7	1	1	2854.2	3	9	2868.6	4	16
C202	2876.5	1	1	2892.9	2	4	3004.5	3	9	3114.1	4	16
C203	2891.8	2	4	2834.1	1	1	3003.5	3	9	3182.4	4	16
C401	5690.6	2	4	5618.6	1	1	6281.4	3	9	6860.3	4	16
C402	4786.4	2	4	4760.3	1	1	5153.2	3	9	5524.5	4	16
C403	6085.2	2	4	6046.4	1	1	6742.1	3	9	6994.5	4	16
R101	2235.9	1	1	2283.2	4	16	2281.5	3	9	2265.4	2	4
R102	1867.5	2	4	1874.3	3	9	1864.2	1	1	2091.7	4	16
R103	2286	1	1	2313.4	4	16	2438.7	4	16	2418.5	3	9
R201	4188.3	1	1	4231.8	2	4	4253.5	3	9	4567.1	4	16
R202	3751.7	1	1	3824.2	2	4	4071.9	3	9	4283.3	4	16
R203	4158.4	1	1	4304.6	3	9	4541.5	4	16	4257.5	2	4
R401	7799.5	1	1	8033.7	3	9	8580.4	3	9	8993.8	4	16
R402	6254	1	1	6559.7	2	4	6893.3	3	9	7417.5	4	16
R403	7872.9	1	1	8129.5	2	4	8338.9	3	9	8933.5	4	16
W101	879.3	1	1	886.3	3	9	893.8	4	16	885.7	2	4
W102	1010.2	2	4	1008.5	1	1	1020.3	3	9	1037.9	4	16
W103	1126.3	1	1	1134.8	2	4	1168.1	3	9	1187.6	4	16
W201	1562.1	2	4	1557.6	1	1	1601.8	3	9	1722.9	4	16
W202	1768.5	1	1	1812.9	2	4	1919.2	3	9	2127.7	4	16
W203	1743.8	1	1	1784.4	2	4	1907.5	3	9	1965.1	4	16
W401	2694.5	1	1	2848.4	2	4	3066.4	3	9	3442.8	4	16
W402	3406	2	4	3360.1	1	1	3692.9	3	9	3984.9	4	16
W501	4896.6	1	1	5056.2	2	4	5224.9	3	9	5858.3	4	16
Avg		1.48			1.96			2.93			3.74	
Sum		40	70		53	125		79	243		101	389

Table 2.5: Ranks among the GGA, the CAC, the ANTS and the SB.

In this comparison, the number of approaches $k = 4$, and the number of instances $b = 27$. Thus, $A_2 = 70+125+243+389 = 827$, while $B_2 = (40^2+53^2+79^2+101^2)/27 = 772.26$.

So the T_2 value is calculated as follows:

$$T_2 = \frac{(27-1)(772.26 - 27\times 4\times 5^2/4)}{827 - 772.26} = 46.2$$

From the F distribution table, with 1-0.01 quantile, $F_{1-\alpha,k-1,(b-1)(k-1)} = F_{0.99,3,78} = 4.04$. Since $T_2 > F_{0.99,3,78}$, we reject the null hypothesis, then there exist at least one approach whose performance is different from at least one of the other metaheuristics.

A paired comparison is then performed to decide which approaches are really different. From a T distribution table, $t_{1-\alpha/2}$ for $\alpha=0.01$ and (b-1)(k-1) = 78 degrees of

freedom is 2.64. Thus, the critical value for the difference is:

$$t_{1-\frac{\alpha}{2}}\left[\frac{2b(A_2-B_2)}{(b-1)(k-1)}\right]^{\frac{1}{2}}=2.64\times\left[\frac{2\times 27\times(827-772.26)}{26\times 3}\right]^{\frac{1}{2}}=16.25$$

Table 2.6 shows the absolute difference of the summation of the ranks between every two approaches. According to the values in the table, the GGA outperforms significantly the ANTS and the SB approaches, but the performance difference between GGA and CAC is not significant.

$\|R_i-R_j\|$	CAC	ANTS [Maniezzo, 2004]	SB [Correia, 2007]
GGA	13	39	61
CAC	-	26	48
ANTS [Maniezzo, 2004]	-	-	22

Table 2.6: Paired comparison results.

The second comparison in chapter 4 is among the AGA, the GGA, the CAC presented in chapter 3 and the ANTS algorithm. The ranks of the solution quality of each approach for each instance are presented in table 2.7.

In this comparison, the number of approaches $k = 4$, and the number of instances $b = 27$. Thus, $A_2 = 35+172+209+394 = 810$, while $B_2 = (29^2+66^2+73^2+102^2)/27 = 775.19$.

So the T_2 value is calculated as follows:

$$T_2=\frac{(27-1)(775.19-27\times 4\times 5^2/4)}{810-775.19}=74.83$$

From the F distribution table, with 1-0.01 quantile, $F_{1-\alpha,k-1,(b-1)(k-1)} = F_{0.99,3,78} = 4.04$. Since $T_2 > F_{0.99,3,78}$, we reject the null hypothesis, then there exist at least one approach whose performance is different from at least one of the other metaheuristics.

A paired comparison is then performed to decide which approaches are really different. From the T distribution table, $t_{1-\alpha/2}$ for $\alpha=0.01$ and (b-1)(k-1) = 78 degrees of freedom is 2.64. Thus, the critical value for the difference is:

$$t_{1-\frac{\alpha}{2}}\left[\frac{2b(A_2-B_2)}{(b-1)(k-1)}\right]^{\frac{1}{2}}=2.64\times\left[\frac{2\times 27\times(810-775.19)}{26\times 3}\right]^{\frac{1}{2}}=12.96$$

Table 2.8 shows the absolute difference of the summation of the ranks between every two approaches. According to the values in the table, the AGA outperforms sig-

nificantly the other approaches.

Instance	AGA			GGA			CAC			ANTS [Maniezzo, 2004]		
	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2	Avg	R	R^2
C101	1585.5	1	1	1599.3	4	16	1593.4	3	9	1592.9	2	4
C102	1704.1	1	1	1712	2	4	1728.2	3	9	1748.5	4	16
C103	1511.6	1	1	1543.9	4	16	1527.5	2	4	1535.1	3	9
C201	2671.5	1	1	2749.4	3	9	2717.7	2	4	2854.2	4	16
C202	2811.9	1	1	2876.5	2	4	2892.9	3	9	3004.5	4	16
C203	2724.6	1	1	2891.8	3	9	2834.1	2	4	3003.5	4	16
C401	5448.9	1	1	5690.6	3	9	5618.6	2	4	6281.4	4	16
C402	4538	1	1	4786.4	3	9	4760.3	2	4	5153.2	4	16
C403	5796.2	1	1	6085.2	3	9	6046.4	2	4	6742.1	4	16
R101	2214.7	1	1	2235.9	2	4	2283.2	4	16	2281.5	3	9
R102	1835.7	1	1	1867.5	3	9	1874.3	4	16	1864.2	2	4
R103	2226.9	1	1	2286	2	4	2313.4	3	9	2438.7	4	16
R201	4117.1	1	1	4188.3	2	4	4231.8	3	9	4253.5	4	16
R202	3666.7	1	1	3751.7	2	4	3824.2	3	9	4071.9	4	16
R203	3982.1	1	1	4158.4	2	4	4304.6	3	9	4541.5	4	16
R401	7405.1	1	1	7799.5	2	4	8033.7	3	9	8580.4	4	16
R402	6222.5	1	1	6254	2	4	6559.7	3	9	6893.3	4	16
R403	7695	1	1	7872.9	2	4	8129.5	3	9	8338.9	4	16
W101	868.2	1	1	879.3	2	4	886.3	3	9	893.8	4	16
W102	1010.5	3	9	1010.2	2	4	1008.5	1	1	1020.3	4	16
W103	1096	1	1	1126.3	2	4	1134.8	3	9	1168.1	4	16
W201	1532	1	1	1562.1	3	9	1557.6	2	4	1601.8	4	16
W202	1739.6	1	1	1768.5	2	4	1812.9	3	9	1919.2	4	16
W203	1696.6	1	1	1743.8	2	4	1784.4	3	9	1907.5	4	16
W401	2608.6	1	1	2694.5	2	4	2848.4	3	9	3066.4	4	16
W402	3273	1	1	3406	3	9	3360.1	2	4	3692.9	4	16
W501	4759.2	1	1	4896.6	2	4	5056.2	3	9	5224.9	4	16
Avg		1.074			2.444			2.704			3.778	
Sum		29	35		66	172		73	209		102	394

Table 2.7: Ranks among the AGA, the GGA, the CAC and the ANTS.

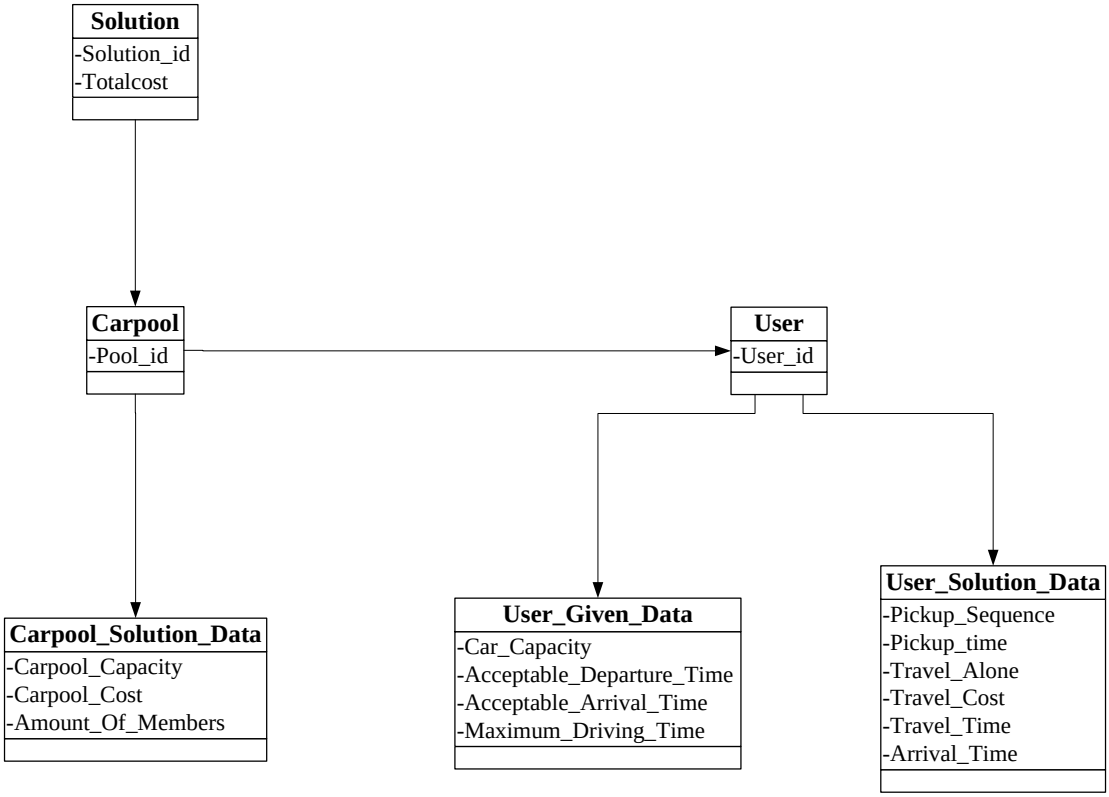
$\lvert R_i - R_j \rvert$	GGA	CAC	ANTS [Maniezzo, 2004]
AGA	37	44	73
GGA	-	7	36
CAC	-	-	29

Table 2.8: Paired comparison results.

.

APPENDIX 3

UML Diagram of the Solution Representation



Publications

The results of the research work presented in this PhD thesis have been published. The publications are listed as follows, grouped by type of publication and sorted chronologically within each category.

Journals

1. Guo Y., Goncalves G., Hsu T. Multi-agent based self-adaptive genetic algorithm for the long-term car pooling problem. International Journal of Mathematical Modeling and Algorithms. Springer, 2012.

2. Guo Y., Goncalves G., Hsu T. A clustering ant colony algorithm for the long-term car pooling problem. International Journal of Swarm Intelligence. IGI Global, 2012.

3. Guo Y., Goncalves G., Hsu T. The multi-destination car pooling problem and its resolution method. RAIRO Operations Research. Cambridge, 2013. (Submitted)

4. Guo Y., Goncalves G., Hsu T. A genetic algorithm with guidance mechanism and adaptive variation rates for solving the long-term car pooling problem. Journal of Operations Research, 2013. (Submitted)

Conferences

5. Guo Y., Goncalves G., Hsu T. A velocity based bee colony algorithm for the car pooling problem. European Conference on Metaheuristics. Lorient, France, 2010.

6. Guo Y., Goncalves G., Hsu T. A self-adaptive genetic algorithm for the car pooling problem. In Proceeding of International Conference on Metaheuristics and Nature Inspired Computing. Djerba, Tunisia, 2010.

7. Guo Y., Goncalves G., Hsu T. Genetic algorithm with preference matrix for the car pooling problem. 12e Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision. Saint Etienne, France, 2011.

8. Guo Y., Goncalves G., Hsu T. A guided genetic algorithm for solving the long-term car pooling problem. In Proceeding of IEEE International Conference on Computational Intelligence in Production and Logistics Systems. Paris, France, 2011.

9. Guo Y., Goncalves G., Hsu T. A clustering ant colony algorithm for the long-term car pooling problem. In Proceeding of IEEE International Conference on Swarm Intelligence. Cergy, France, 2011.

10. Guo Y., Goncalves G., Hsu T. An ant colony algorithm based hybrid approach for solving the multi-destination car pooling problem. 13e Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision. Angers, France, 2012

Bibliography

[Baldacci et. al., 2004] Baldacci R., Maniezzo V., Mingozzi A. (2004). An exact method for the car pooling problem based on Lagrangian column generation. Operational Research, 52(3), 422-439.

[Beni and Wang, 1989] Beni G., Wang J. (1989). Swarm intelligence in cellular robotic systems. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy.

[Berger et al., 2003] Berger J., Barkaoui M., Bräysy O. (2003). A route-directed hybrid genetic approach for the vehicle routing problem with time windows. Information Systems and Operational Research, 41, 179-194.

[Blum and Roli, 2003] Blum C., Roli A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computer Survey, 35(3), 268–308.

[Blumenthal et. al., 1997] Blumenthal C., Michalak S., Goble B., Garner M., Haselkorn M., Spyridakis J. (1997). Bellevue smart traveler: Design, demonstration and assessment, Internet, USA.

[Bodin and Sextion, 1986] Bodin L.D., Sexton T. (1986). The multi-vehicle subscriber dial-a-ride problem. TIMS Studies in Management Science, 2, 73-86.

[Bräysy, 2003] Bräysy O. (2003). A reactive variable neighborhood search for the vehicle routing problem with time windows. INFORMS Journal on Computing, 15, 347 -368.

[Brotcorne et al., 2003] Brotcorne L., Laporte G., Semet F. (2003). Ambulance location and relocation models. European Journal of Operational Research, 147(3), 451-463.

[Burke et al., 2009] Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J. (2009). A classification of hyper-heuristics approaches. Computer Science Technical Report, No.NOTTCS-TR-SUB- 6490907061259-5808.

[Calvo and Colorni, 2007] Calvo R.W., Colorni A. (2007). An approximation algorithm for the dial-a-ride problem. Journal of Operations Research, 5(1), 61-73.

[Calvo et. al., 2004] Calvo R.W., Luigib F., Haastrupc P., Maniezzod V. (2004). A distributed geographic information system for the daily car pooling problem. Computers and Operations Research archive, 31(13), 2263-2278.

[Cerny, 1985] Cerny V. (1985). Thermo dynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45, 41-51.

[Conover, 1998] Conover W. (1998). Practical nonparametric stactistics. Wiley, New York, USA.

[Cordeau and Laporte, 2003] Cordeau J.F., Laporte G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part

B: Methodological, 37(6), 579-594.

[Cordeau et al., 2001] Cordeau, J.F., Laporte G., Mercier A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. Journal of Operational Research, 52, 928–936.

[Correia and Viegas, 2008] Correia G., Viegas J. (2008). Structured simulation-based methodology for carpooling viability assessment. Transportation Research Board 87th Annual Meeting, Washington DC, USA.

[Cramer, 1985] Cramer N. L. (1985). A representation for the adaptive generation of simple sequential programs. In Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Erlbaum, 183-187.

[Cung et al., 1997] Cung V-D., Mautor T., Michelon P., Tavares A. (1997). A scatter search based spproach for the quadratic assignment problem. In Proceedings of IEEE International Conference on Evolutionary Computation, Indianapolis, USA, 165-170.

[Cung et al, 2001] Cung V-D., Martins S.L., Ribeiro C.C., Roucairol C. (2001). Strategies for the parallel implementation of metaheuristics. Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, 263-308.

[De Backer and Furnon, 1997] De Backer B., Furnon V. (1997). Meta-heuristics in constraint programming experiments with tabu search on the vehicle routing problem. In Proceeding of Second International Conference of Metaheuristics, Sophia Antipolis, France, 1-14.

[Denzinger et al., 1997] Denzinger J., Fuchs M., Fuchs, M. (1997). High performance ATP systems by combining several AI methods. In Proceeding of the 15th International Joint Conference on Artificial Intelligence, 102–107.

[Desrosiers et al., 1986] Desrosiers J., Dumas Y., Soumis F. (1986). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. American Journal of Mathematical and Management Sciences, 6, 301-325.

[Dorigo and Stützle, 2004] Dorigo M., Stützle T. (2004). Ant Colony Optimization. MIT Press.

[Dorigo et al., 1996] Dorigo M., Maniezzo V., Colorni A. (1996). Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Part B, 26(1), 29-41.

[Dorigo, 1992] Dorigo M. (1992). Optimization, learning and natural algorithms, PhD Thesis. Politecnico di Milano, Italie.

[Dumas et al., 1989] Dumas Y., Desrosiers J., Soumis F. (1989). Large scale multi-vehicle dial-a-ride problems. Les Cahiers du GERAD, École des Hautes Études Commerciales, Canada.

[Feo and Resende, 1989] Feo T.A., Resende M.G.C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 8, 67–71.

[Ferrari and Manzini, 2003] Ferrari E., Manzini R. (2003). The car pooling problem: heuristic algorithms based on savings functions. Journal of Advanced

Transportation, 37(3), 243-272.

[Fisher, 1981] Fisher M.L. (1981). The Lagrangian relaxation method for solving integer programming problem. Management Science, 27, 1-18.

[Fogel et al., 1966] Fogel L.J., Owens A.J., Walsh M.J. (1966). Artificial Intelligence Through Simulated Evolution. John Wiley & Sons, New York, USA.

[Fraser and Burnell, 1970] Fraser A., Burnell D. (1970). Computer models in genetics. New York: McGraw-Hill.

[Friedman, 1940] Friedman M. (1940). A comparison of alternative tests of significance for the problem of m rankings. The Annals of Mathematical Statistics 11, 1, 86–92.

[Gambardella et al., 1999] Gambardella L.M., Taillard E., Agazzi G. (1999). MACS- VRPTW: A multiple ant colony system for vehicle routing problems with time windows. New Ideas in Optimization. McGrawHill, London, UK, 63-76.

[Garcia et al., 1994] Garcia B.L., Potvin J.Y., Rousseau J.M. (1994). A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. Computers & Operations Research, 21, 1025-1033.

[Gehring and Homberger, 2001] Gehring H., Homberger J. (2001). Parallelization of a two-phase metaheuristic for routing problems with time windows. Asia-Pacific Journal of Operational Research, 18, 35-47.

[Glover, 1986] Glover F. (1986). Future paths for integer programming and links to artificial intelligence. Computer and Operational Research, 13, 533-549.

[Glover, 1990] Glover F. (1990). Tabu search: A tutorial. Interfaces.

[Goldberg, 1989] Goldberg D.E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Professional, 1 edition.

[Hansen and Mladenovic, 1997] Hansen P., Mladenovic N. (1997). Variable neighborhood search for the p-median. Location Science, 5, 207-226.

[Hansen and Mladenovic, 1999] Hansen P., Mladenovic N. (1999). Variable neighborhood search: Principles and applications. European Journal of Operational Research, 130(3), 449-467.

[Healy and Moll, 1995] Healy P., Moll R. (1995). A new extension of local search applied to the dial-a-ride problem. European Journal of Operational Research, 83(1), 83-104.

[Hyde, 1999] Hyde P. (1999). Java thread programming. Sams publishing.

[Ioachim et al., 1995] Ioachim I., Desrosiers J., Dumas Y., Solomon M.M. (1995). A request clustering algorithm for door-to-door handicapped transportation. Transportation Science, 29, 63-78.

[Jaw et al., 1986] Jaw J., Odoni A.R., Psaraftis H.N., Wilson N.H.M. (1986). A heuristic algorithm for the multi-vehicle advance-request dial-a-ride problem with time windows. Transportation Research, B20, 243-257.

[Jedrzejowicz and Wierzbowska, 2006] Jedrzejowicz P., Wierzbowska I. (2006).

Jade-based A-team environment. In Proceeding of the 6th International Conference on Computational Science, 28–31.

[Jung and Moon, 2002] Jung S., Moon B.R. (2002). A hybrid genetic algorithm for the vehicle routing problem with time windows. In Proceeding of Genetic Evolutionary Computation Conference, Morgan Kaufmann, San Francisco, USA, 1309-1316.

[Kallehauge et al., 2005] Kallehauge B., Larsen J., Madsen B.G., Solomon M.M. (2005). Vehicle routing problem with time windows. Column Generation, 67-98.

[Karaboga, 2010] Karaboga D. (2010). Artificial bee colony algorithm. Scholarpedia, 5(3): 6915.

[Kennedy and Eberhart, 1995] Kennedy J., Eberhart R. (1995). Particle swarm optimization. In Proceedings of IEEE International Conference on Neural Networks IV, 1942-1948.

[Kirkpatrick et al., 1983] Kirkpatrick S., Gelatt C. D., Vecchi M. P. (1983). Optimization by simulated annealing. Science, New Series, 220(4598), 671-680.

[Kothari, 2004] Kothari A.B. (2004). Genghis: a multi-agent carpooling system. Dissertation, University of Bath, UK.

[Lau et al., 2003] Lau H.C., Sim M., Teo K.M. (2003). Vehicle routing problem with time windows and a limited number of vehicles. European Journal of Operational Research, 148, 559-568.

[Li and Lim, 2003] Li H.; Lim A. (2003). A metaheuristic for the pickup and delivery problem with time windows. International Journal on Artificial Intelligence Tools, 12(2), 173-186.

[Li et al., 2003] Li H., Lim A., Huang J. (2003). Local search with annealing-like restarts to solve the VRPTW. European Journal of Operational Research, 150, 115 -127.

[MacQueen, 1967] MacQueen J.B. (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, 281-297.

[Maniezzo et al., 2004] Maniezzo V., Carbonaro A., Vigo D., Hildmann H. (2004). An ANTS algorithm for the long-term car pooling problem. New Optimization Techniques in Engineering, 15, 411-430.

[Maurizio et al., 2011] Maurizio B., Diego C., Alberto C., Alessandro L. (2011). PoliUniPool: a car pooling system for universities. Social and Behavioral Sciences, 20, 558-567.

[Mester, 2002] Mester D. (2002). An evolutionary strategies algorithm for large scale vehicle routing problem with capacitate and time window restrictions. Working paper, Institute of Evolution, University of Haifa, Israel.

[Meyers et. al., 1999] Meyers D., Dailey D.J., Lose D. (1999). Seattle smart traveler: dynamic ride matching on the World Wide Web. Transportation Research C, 7, 17-32.

[Miller and Goldberg, 1995] Miller B.L., Goldberg D.E. (1995). Genetic algorithms, tournament selection and the effects of noise. Complex Systems, 9, 193-212.

[Potvin and Bengio, 1996] Potvin J.Y., Bengio S. (1996). The vehicle routing problem with time windows, Part II: Genetic search. INFORMS Journal on Computing, 8, 165-172.

[Psaraftis, 1980] Psaraftis H.N. (1980). A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem. Transportation Science, 14, 130-154.

[Russell and Norvig, 2003] Russell J., Norvig P. (2003). Artificial Intelligence: A modern approach. upper saddle river. New Jersey, Prentice Hall, 111-114.

[Russell, 1995] Russell R. A. (1995). Hybrid heuristics for the vehicle routing problem with time windows. Transportation Science, 29, 156-166.

[Schwefel, 1981] Schwefel H.P. (1981). Numerical Optimization for Computer Models. John Willey, Chichester, U.K..

[Sexton and Bodin, 1985a] Sexton T., Bodin L.D. (1985). Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. Transportation Science, 19, 378-410.

[Sexton and Bodin, 1985b] Sexton T., Bodin L.D. (1985). Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. Transportation Science, 19, 411-435.

[Solomon, 1987] Solomon M.M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. Operational Research, 35, 254-265.

[Souvaine, 2008] Souvaine D. (2008). Line segment intersection using a sweep line algorithm. Computational Geometry, Tufts University.

[Taillard et al., 1997] Taillard E., Badeau P., Gendreau M., Guertin F., Potvin J.Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science, 31, 170-186.

[Talbi and Bachelet, 2004] Talbi E.G., Bachelet V. (2004). Cosearch: a parallel co-evolutionary metaheuristic. In Proceeding of Workshop on Hybrid Metaheuritics, 127–140.

[Tan et al., 2000] Tan K.C., Lee L.H., Zhu K.Q. (2000). Heuristic methods for vehicle routing problem with time windows. In Proceeding of 6th International Symposium. of Artificial Intelligence, FL.

[Tan et al., 2001a] Tan K.C., Lee L.H., Ou K. (2001). Hybrid genetic algorithms in solving vehicle routing problems with time window constraints. Asia-Pacific Journal of Operational Research, 18, 121-130.

[Tan et al., 2001b] Tan K.C., Lee L.H., Ou K. (2001). Artificial intelligence heuristics in solving vehicle routing problems with time window constraints. Engineering Applications of Artificial Intelligence, 14, 825-837.

[Thangiah et al., 1995] Thangiah S.R., Osman I.H., Vinayagamoorthy R., Sun T. (1995). Algorithms for the vehicle routing problems with time deadlines. American Journal of Mathematical and Management Sciences, 13, 323-355.

[Toth and Vigo, 1997] Toth P., Vigo D. (1997). Heuristic algorithms for the handicapped person's transportation problem. Transportation Science, 31, 60-71.

[Vargas et al., 2008] Vargas M., Sefair J., Walteros J., Medaglia A.L., Rivera L. (2008). Car pooling optimization: a case study in Strasbourg. Systems and Information Engineering Design Symposium, Virginia, USA.

[Varrentrapp et al., 2002] Varrentrapp K., Maniezzo V., Stützle T. (2002). The long term car pooling problem on the soundness of the problem formulation and proof of NP-completeness. Technische Universitat Darmstadt, Germany.

[Villegas, 2011] Villegas J.G. (2011). Using nonparametric tests to compare the performance of metaheuristics. EU/ME - The metaheuristics community, internet.

[Wee Kit et al., 2001] Wee Kit H., Chin J., Lim A. (2001). A hybrid search algorithm for the vehicle routing problem with time windows. International Journal on Artificial Intelligence Tools, 10, 431-449.

[Wilson et. al., 1971] Wilson H., Sussman J., Wang H., Higonnet B. (1971) Scheduling algorithms for dial-a-ride system. Urban Systems Laboratory, MIT, USA.

[Yan et al., 2011] Yan S., Chen C., Lin Y. (2011). A model with a heuristic algorithm for solving the long-term many-to-many car pooling problem. IEEE Transactions on Intelligent Transportation Systems, 14(4), 1362-1373.

[Yan, 1996] Yan S. (1996). Approximating reduced costs under degeneracy in a network flow problem with side constraints. Networks, 27, 267-278.