VOT 74285

DEVELOPMENT OF HEURISTIC METHODS BASED ON GENETIC ALGORITHM (GA) FOR SOLVING VEHICLE ROUTING PROBLEM

(PEMBANGUNAN KAEDAH HEURISTIK BERASASKAN ALGORITMA GENETIK UNTUK MENYELESAIKAN MASALAH PENJALANAN KENDERAAN)

ZUHAIMY BIN ISMAIL

RESEARCH VOTE NO :
74285

JABATAN MATEMATIK
FAKULTI SAINS
UNIVERSITI TEKNOLOGI MLAYSIA

2008

ABSTRACT

DEVELOPMENT OF HEURISTICS METHODS BASED ON GENETIC ALGORITHM FOR VEHICLE ROUTING PROBLEM (VRP)

The Vehicle Routing Problem (VRP) is an important area and has been studied as combinatorial optimization problems. VRP calls for the determination of the optimal set of routes to be performed by a fleet of vehicle to serve a given set of customers. VRP in which demand at each location is unknown at the time when the route is designed but is follow a known probability distribution, is known as VRP with Stochastic Demands (VRPSD). VRPSD finds its application on wide-range of distribution and logistics-transportation sector with the objective is to serve a set of customers at minimum total expected cost. One of the applications of VRPSD is in the case of picking up garbage done by solid waste collection company.

The computational complexity of most vehicle routing problem and moreover the intricate of stochastic VRP algorithm has made them an important candidate for solution using metaheuristics. This research proposes the enhanced metaheuristic algorithms that exploit the power of Tabu Search, Genetic Algorithm, and Simulated Annealing for solving VRPSD. Genetic Algorithm as population-based methods are better identifying promising areas in the search space, while Tabu Search and Simulated Annealing as trajectory methods are better in exploring promising areas in search space.

Simulated Annealing is a global optimization technique which traverses the search space by generating neighboring solutions of the current solution. A superior neighbor is always accepted and an inferior neighbor is accepted with some probability. Tabu Search is similar to Simulated Annealing, in that both traverse the solution space by testing mutations of an individual solution. However, simulated annealing generates only one mutated solution but Tabu Search generates many mutated solutions and moves to the solution with the lowest fitness of those generated. Genetic Algorithm gives a pool of solutions rather than just one. The process of finding superior solutions mimics the evolution process, with solutions being combined or mutated to find out the pool of solutions. This research explored and developed new heuristics based on GA for solving VRPSD. New algorithms, journal papers and computerized system were also developed. Future, area that may be explored include the used of Ant Colony Optimization (ACO) which exploits the nature phenomenon of ants. Based on the proposed heuristic method, we developed a program to optimize the routing problem using the Visual Studio C++ 6.0 programming language.

Key researchers :

Assoc. Prof. Dr. Zuhaimy bin Haji Ismail
Irhamah Nurhadi
Dr Zaitul Marlizawati Zainuddin

E-mail : zuhaimyi@yahoo.com
Tel. No : 07-5534224
Vote No : 74285

ABSTRAK

PEMBANGUNAN KAEDAH HEURISTIK BERASASKAN KAEDAH GENETIC ALGORITMA (GA) UNTUK MASALAH PERJALANAN KENDERAAN (VRP).

Masalah Perjalanan Kenderaan (VRP) adalah satu ruang yang penting untuk dikaji sebagai salah satu daripada gabungan masalah pengoptimaan. VRP juga dikenali sebagai pencarian suatu set optimum kepada perjalaan kenderaan untuk mewakili atau di anggap sebagai satu set pelanggan. VRP untuk setiap permintaan dan disetiap lokasi adalah satu anu yang tidak diketahui ketika perjalanan dibentuk, akan tetapi ianya akan mengikut setiap taburan kebarangkalian yang diketahuinya, dan ini juga dikenali sebagai VRP bersama permintaan Stokastik (VRPSD). Aplikasi VRPSD ditemui dalam suatu ruang kebarangkalian yang luas dan sektor pengangkutan logistik bersama objektifnya yang tersendiri iaitu untuk menghasilkan set pelanggan pada kadar minimum. Salah satu daripada aplikasi VRPSD adalah daripada kes untuk mengangkut bahan buangan pepejal yang telah dilakukan oleh syarikat pengangkutan bahan buangan pepejal.

Kompleks Pengkomputeran adalah masalah system pengangkutan kenderaan yang lebih kompleks kepada algoritma stokastik. *VRP* telah membuatkan ianya sebagai satu cara yang penting untuk penyelesaian masalah menggunakan kaedah metaheuristik. Kajian ini bertujuan untuk memberi penekanan yang lebih mendalam kepada kaedah metaheuristik yang membentuk kaedah lain, seperti *Tabu Search*, *Genetic Algorithm*, dan *Simulated Annealing* untuk menyelesaikan masalah VRPSD. Algoritma Genetic adalah kaedah populasi asalan sebagai suatu kaedah yang baik dalam mencari identiti di ruang carian, akan tetapi *Tabu Search* dan *Simulated Annealing* adalah kesinambungan yang lebih baik di ruang carian.

Simulated Annealing adalah teknik optimum global yang telah melangkaui ruang carian dengan menghasilkan kebanyakan penyelesaian untuk masalah berjiran. Suatu set berjiran yang matang dan jiran yang berdekatan akan sentiasa diterima dengan apa sahaja kemungkinan. Tabu Search mempunyai ciri-ciri yang sama seperti *Simulated Annealing*, kedua-duanya menguasai penyelesaian ruang untuk pencubaan mutasi daripada masalah individu. Walaubagaimanapun, simulated annealing menghasilkan hanya satu penyelesaian mutasi, manakala tabu search menghasilkan lebih daripada satu masalah mutasi, dan membentuk penyelessaian tersebut kepada bentuk penghasilan yang terbaik. Genetic Algorithm memberi lebih daripada satu penyelesain, malah satu koleksi kumpulan penyelesaian. Evolusi proses, iaitu suatu proses mencari ajukan kepada *superior solutions* bersama penyelesaiannya, telah dikombinasikan atau dimutasikan, untuk mencari satu kumpulan penyelesaian. Kajian ini diselidik dan telah membentuk satu kaedah baru untuk heuristics yang berasaskan GA bagi menyelesaikan masalah VRPSD. Algoritma-algoritma baru, akhbar jurnal dan sistem perkomputeran juga telah dihasilkan. Tambahan lagi, bahagian yang dikaji juga bersangkutan dengan kaedah yang digunkan untuk *Ant Colony Optimization (ACO)* yang telah dibentuk mengikut fenomena semut. Merujuk kepada tujuan kaedah heuristic, dengan ini kami membentuk suatu program untuk mengoptimumkan masalah perjalanan dengan menggunakan Visual Studio C++ 6.0 programming.

Key researchers :

Assoc. Prof. Dr. Zuhaimy bin Haji Ismail
Irhamah Nurhadi
Dr Zaitul Marlizawati Zainuddin

E-mail	:	zuhaimyi@yahoo.com
Tel. No	:	07-5534224
Vote No	:	74285

UNIVERSITI TEKNOLOGI MALAYSIA

BORANG PENGESAHAN
LAPORAN AKHIR PENYELIDIKAN

TAJUK PROJEK : DEVELOPMENT OF HEURISTIC METHODS BASED ON GENETIC ALGORITHM FOR SOLVING VEHICLE ROUTING PROBLEM

Saya ZUHAIMY BIN ISMAIL
(HURUF BESAR)

Mengaku membenarkan **Laporan Akhir Penyelidikan** ini disimpan di Perpustakaan Universiti Teknologi Malaysia dengan syarat-syarat kegunaan seperti berikut :

1. Laporan Akhir Penyelidikan ini adalah hakmilik Universiti Teknologi Malaysia.

2. Perpustakaan Universiti Teknologi Malaysia dibenarkan membuat salinan untuk tujuan rujukan sahaja.

3. Perpustakaan dibenarkan membuat penjualan salinan Laporan Akhir Penyelidikan ini bagi kategori TIDAK TERHAD.

4. * Sila tandakan (/)

☐ SULIT (Mengandungi maklumat yang berdarjah keselamatan atau Kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972).

☐ TERHAD (Mengandungi maklumat TERHAD yang telah ditentukan oleh Organisasi/badan di mana penyelidikan dijalankan).

[/] TIDAK TERHAD

TANDATANGAN KETUA PENYELIDIK

PROF. MADYA. DR. ZUHAIMY ISMAIL
Nama & Cop Ketua Penyelidik

Tarikh : 1 MARCH 2008

CATATAN : * *Jika Laporan Akhir Penyelidikan ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali sebab dan tempoh laporan ini perlu dikelaskan sebagai SULIT dan TERHAD.*

PREFACE

This research report entitled “The development of heuristic methods based on Genetic Algorithm (GA) for solving Vehicle Routing Rroblem (VRP)” as been prepared by Assoc. professor Dr. Zuhaimy Hj. Ismail during the period January 2006 to February 2008 at the Department of Mathematics, University Technology Malaysia, Skudai Johor.

This report is submitted as the requirement for the completion of e-science research project which is fully supported by The Ministry of Science, Technology and Innovation (MOSTI). The subject of this report is to develop heuristic methods for solving variant of VRP basing the heuristics on GA. One such variant is the VRP with stochastic demand (VRPSD). I would like to express my gratitude to MOSTI for their trust in our expertise and interest in this project.

Next, I would like to thank The Research management Centre (RMC), Universiti Teknologi Malaysia for all the support and services provided in making this research a success. I would like to thank The Department of Mathematics and Faculty of Science for making this research a success.

I would like to thank Mr. Ahmad Kamel Ismail from The SWM Environment Sdn Bhd (formerly known as Southern Waste Management Sdn Bhd) and the management of Perniagaan Zawiyah Sdn. Bhd for our collaboration on the data collection and for providing the guidelines on the processes in solid waste collection and distribution practices.

I also wish to thank my students and colleagues Irhamah, Khairil Asmani, Dr Zaitul Marlizawati for their contribution and devotion in this work. Also. The staff – the academic as well as the administrative – at The Department of Mathematics, Faculty of Science, UTM

Assoc. Prof. Dr. Zuhaimy Ismail
ISDAG and ISORG, Fakulti Sains, UTM

SUMMARY

The Vehicle Routing Problem (VRP) is an important area and has been studied as combinatorial optimization problems. VRP calls for the determination of the optimal set of routes to be performed by a fleet of vehicle to serve a given set of customers. VRP in which demand at each location is unknown at the time when the route is designed but is follow a known probability distribution, is known as VRP with Stochastic Demands (VRPSD). VRPSD finds its application on wide-range of distribution and logistics-transportation sector with the objective is to serve a set of customers at minimum total expected cost. One of the applications of VRPSD is in the case of picking up garbage done by solid waste collection company.

The computational complexity of most vehicle routing problem and moreover the intricate of stochastic VRP algorithm has made them an important candidate for solution using metaheuristics. This research proposes the enhanced metaheuristic algorithms that exploit the power of Tabu Search, Genetic Algorithm, and Simulated Annealing for solving VRPSD. Genetic Algorithm as population-based methods are better identifying promising areas in the search space, while Tabu Search and Simulated Annealing as trajectory methods are better in exploring promising areas in search space.

Simulated Annealing is a global optimization technique which traverses the search space by generating neighboring solutions of the current solution. A superior neighbor is always accepted and an inferior neighbor is accepted with some probability. Tabu Search is similar to Simulated Annealing, in that both traverse the solution space by testing mutations of an individual solution. However, simulated annealing generates only one mutated solution but tabu search generates many mutated solutions and moves to the solution with the lowest fitness of those generated. Genetic Algorithm gives a pool of solutions rather than just one. The process of finding superior solutions mimics the evolution process, with solutions being combined or mutated to find out the pool of solutions. This research explored and developed new heuristics based on GA for solving VRPSD. New algorithms, journal papers and computerized system were also developed. Future, area that may be explored include the used of Ant Colony Optimization (ACO) which exploits the nature phenomenon of ants. Based on the proposed heuristic method, we developed a program to optimize the routing problem using the Visual Studio C++ 6.0 programming language.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Optimization is a part of life. In our day to day lives, we make decisions that we believe can maximize or minimize our objectives, such as taking a shortcut to minimize the time or distance required to reach a particular destination, or finding a lowest priced items in the supermarket. Most of these decisions are based on our years of experience and knowledge about the system without resorting to any systematic mathematical formulation. However, as the system becomes more complicated, further it is needed to formulate it into specific mathematical model, and with the advent of computer it is possible to exploit optimization theories to their maximum extent.

Combinatorial optimization is a branch of optimization that arises everywhere and certainly in applied mathematics and computer science, related to operations research, algorithm theory and computational complexity theory that sit at the intersection of several fields, including artificial intelligence, mathematics and software engineering. Combinatorial optimization algorithms solve problem instances that are believed to be hard in general, by exploring the usually large solution space of these instances. Combinatorial optimization algorithms achieve this by reducing the effective size of the space and by exploring the space efficiently.

The Vehicle Routing Problems (VRP), Traveling Salesman Problem (TSP), minimum spanning tree problem and knapsack problem are example problems of combinatorial optimization.

Since late fifties, the Vehicle Routing Problem (VRP) has been and remains a rich topic for researchers and practitioners. It becomes an area of importance to operations research as well as its use for real world applications. An integral component of logistics is transportation, and a frequently arising situation in the transportation and distribution of commodities has usually been modeled as a Vehicle Routing Problem (VRP). Usually real world VRP arises with many site constraints. VRP is a generalized problem of the Traveling Salesman Problem (TSP) in that the VRP consists in determining *m* vehicle, where a route is tour that begins at the depot. The task is to visit a set of customer in a given order and returns to the depot. All customers must be visited exactly once and the total customer demand of a route must not exceed the vehicle capacity. Given a set of geographically dispersed customers, each showing a positive demand for a given commodity, the VRP consists of finding a set of tours of minimum length (or cost) for a fleet of vehicles.

The majority of these researches conducted on operations research are focus on static and deterministic cases of vehicle routing in which all information is determined before the time of planning of the routes. Whereas in this ICT age, information is gathered in real time and in many cases they are changing. The complexity of the problem increases as more information is unavailable at the time of the planning or when the service has begun such as the time to begin service, the location, actual demand. In some service industries, it allow for customers to request for service within a short period of time and such request has increase the dynamism and complexity of the problem.

In most real life application, stochastic or dynamic information occurs parallel to the routes being carried out. Many of the vehicle routing problems have inherent randomness, which is not considered in deterministic models, probably travel times or demands are random variables with known distributions. The work by Tillman (1969) was the first that has brought us to explore cases on VRP with stochastic demands. Since that, many theories and algorithms on VRPSD have been proposed and or developed. In this thesis, we interest in studying variables as stochastic part. This chapter presents the flow of the research proposal and it begins with the background and problem statement of the research. It is important that an extensive work has been carried out in order to present a case for this work and this is given in Chapter 2. Research objectives, the scope of this study and discussion on the research contribution are also given. Finally, the brief of each chapter is outlined.

1.2 Background of the problem

The classical VRP models usually do not capture an important aspect of real life transportation and distribution-logistic problems, namely fact that several of the problem parameters (demand, time, distance, city location, etc) are often stochastic. Most existing VRP models oversimplify the actual system by assuming system parameter (e.g. customer demands) as deterministic value, although in real application, it may not be possible to know all information about customers before designing routes. Stochastic information occurs and has major impact on how the problem is formulated and how the solution is implemented. As compared to the development in deterministic case, research in Stochastic VRP is rather undeveloped.

This study considers an important variation of VRP that is VRP with Stochastic Demands (VRPSD) in which demand at each location is unknown at the time when the route is designed, but is follow a known probability distribution. This situation arises in practice when whenever a company, on any given day, is faced with the problem of collection/ deliveries from or to a set of customers, each has a random demand. In this study, we deal with specific case at solid waste collection. It is hoped that optimization can take into account the stochasticity of the problem in obtaining better routes or reducing cost.

In stochastic environment, due to its randomness in customers' demands, a vehicle capacity may be exceeded during service. A route failure is said to occur if the demand exceeds capacity and a recourse action needs to be taken. Assuming that enough capacity is available at the depot, the vehicle may return to the depot, replenish its load, and then resume service at the point where failure occurred. Therefore the vehicle will always be able to satisfy all demands, and the length of the corresponding tour becomes a random quantity. The recourse action could be the vehicle resumes service along the planned route, namely *a priori* approach (Bertsimas et al., 1990)*,* or visiting the remaining customers possibly in an order that differs from the planned sequence that is called *re-optimization* approach (Dror et al, 1989).

Tillman (1969) was the first to propose algorithm for the VRPSD in the case where there were multiple terminal deliveries and multiple vehicles. Since then, many researchers have studied this problem in two frameworks, namely the chance constrained and stochastic with recourse. In the chance constrained VRPSD, the problem is to design a number of vehicle routes of least distance traveled, subject to constraint that the probability of route failure on any route is within an allowable limit. In contrast, VRPSD with recourse try to minimize the total expected cost (or distance), including the cost of travel as well as the cost of recourse action when a route failure occurs. The VRPSD with recourse is considerably more difficult than chance constrained VRPSD (Yang et al., 2000).

Various formulations and algorithms have been proposed and investigated, including the properties and solution frameworks of VRPSD studied by Dror et al. (1989), Bertsimas (1992) who proposed cyclic heuristic and found a priori solution for single vehicle and Dror et al. (1993) who have examined a priori VRPSD in the context of Stochastic Programming where there is only one vehicle and the number of potential failures is small. Yang's thesis (1996) developed optimal restocking policy in conjunction with routing decisions for a priori VRPSD for single and multiple vehicles. Secomandi (2001) considered re-optimization-type routing policy by means of rollout policy for single vehicle. Chepuri and Homem-de-Mello (2005) proposed a new heuristic method based on the Cross-Entropy method for single vehicle.

Bianchi et al. (2005) considered basic implementation of five metaheuristics for single vehicle: Iterated Local Search, Tabu Search, Simulated Annealing, Ant Colony Optimization and Evolutionary Algorithm (Genetic Algorithm) that found better solution quality in respect to cyclic heuristic. Instead of the work of Bianchi et al. (2005) and Gendreau et al. (1995), the work on the application of GA, TS and ACO for VRPSD are lacking in the literature, although it is widely known that GA has been proven effective and successful in a wide variety of combinatorial optimization problems, including certain types of VRP, especially where time windows are included and in TSP, and as known also that Tabu Search, the approach that dominates the list of successful algorithms that is a robust, efficient and effective approach to the general VRP family of problem (Laporte 1992, Osman 1993) and Tabu Search often outperforms other heuristic techniques in terms of computational speed and solution quality (Osman, 1993).

The number of published work on the application of GA for solving basic VRP, TSP, VRPTW, VRPB, and multi depot VRP has been growing. Different approaches were also proposed based on different crossover operator, different mutation operator, or replacement methods. In 2005, the work of Bianchi et al. results that the performance of GA and TS seem to be not significantly different, due to the fact that these algorithms find solutions values which are not very different to each other. Based on previous research on algorithm developed for VRPSD and the knowledge of the basic structure of GA and TS, in this study we develop a TS and GA for single VRPSD in the next chapter.

Although pure GA performs well, mostly it does not equal Tabu Search in terms of solution quality, sometimes pure GA perform inefficient on practical combinatorial optimization. To improve pure GA performance, some algorithms are combined with the simple GA, yielding a hybrid algorithm. The statement about GA hybridization is noted by Coley (1999) that hybrid algorithms, which combine a GA with more traditional algorithms, have been hinted as a highly powerful combination for solving practical problem, also by Lacomme et al. (2005) that it is well known that a standard GA must be hybridized with another search procedure to be able to compete with metaheuristics like Tabu Search. Baker and Ayechew (2003) showed that hybrid GA with neighbourhood search in the basic VRP is competitive with tabu search and simulated annealing in terms of solution time and quality. Hybrid Genetic Algorithms also have widespread application to VRPTW, including the work of Blanton and Wainwright (1993), Thangiah (1995a and 1995b), Berger et al. (1998) and Braysy et al. (2000).

In this study, several metaheuristics were developed for solving VRPSD, including GA, TS and ACS. As known, GA and ACS as population-based methods are better in identifying promising areas in the search space, whereas TS as trajectory methods are better in exploring promising areas in search space.

1.3 Problem Statement of Research

The VRPSD is defined on a complete graph $G = (V, A, C)$, where

V = $\{0, 1, \ldots, n\}$ is a set of nodes with node 0 denotes the depot and nodes 1, 2, …, n correspond to the customers,

A = $\{(i, j) : i, j \in V, i \neq j\}$ is the set of arcs joining the nodes,

and a non-negative matrix

C = ($c_{ij} : i, j \in V, i \neq j\}$) denotes the travel cost (distance) between node i and j.

The cost matrix C is symmetric and satisfies the triangular inequality. Customers have stochastic demands ξ_i, $i = 1, \ldots, n$, which are a non-negative discrete random variable with known probability distribution $p_{ik} = \Pr ob(\xi_i = \xi^k)$, $k = 0, 1, \ldots, K \leq Q$. Assume further that customers' demands are independent and identical. Actual demand is only known after the vehicle arrives at customer's location. If there is a route failure at any node, the recourse action has to be made, the recourse action is travels back to the depot for replenish and then to resume its journey as planned at the node where failure occurred.

A stochastic vehicle routing problem arises when not all information relevant to the planning of the routes is known by the planner when the routing process begins and information can change after the initial routes have been constructed. Algorithms for stochastic VRP are considerably more intricate than deterministic and it calls for efficient algorithm that is able to work in real-time since the immediate requests should be served. These have made them an important candidate for solution using metaheuristics.

This research tries to propose new metaheuristic algorithm based on hybrid Genetic Algorithm with Tabu Search to solve VRPSD. In this work, we initially consider a single-vehicle model and later expand our analysis to incorporate multiple homogeneous and heterogeneous vehicles. In the hybrid GA, Tabu Search is used as mutation operator. The performance of this algorithm will be compared with other heuristics/ metaheuristics and implementation of this algorithm also will be done to solve real problem in optimizing solid waste collection.

1.4 Objectives of the Study

The aim of this study is to develop various approaches to optimize VRPSD solution, particularly in the use of metaheuristics approach. The objectives of this study are to:

a. develop metaheuristics for solving single VRPSD that include:
   1. Tabu Search
   2. Genetic Algorithm
   3. Ant Colony Optimization
   4. Simulated Annealing

b. conduct comparative evaluation on the performance of the above metaheuristics.

c. implement the VRPSD models for solving real problem data in optimizing solid waste collection.

1.5 Scope of the Study

In this research, we confine the application of our algorithm to randomly generated problem following some discrete probability distributions for the performance testing, and also implementing the algorithm to solve real problem data in optimizing solid waste collection.

The scope of this study can be summarized as follows:

1. It is assumed that vehicles are start and end at a single depot.
2. It is also assumed that a customer demand can not be split between vehicles, and that the demand is to be fully satisfied.
3. The recourse action to be taken is a priori approach.
4. The problem data were generated and tested using Minitab 14 and SPSS 13, while the implementation of metaheuristics was done using Delphi 7.0.

1.6 Significance of the Study

From the view point of VRP, our algorithm is the first implementation of hybrid GA (especially hybrid GA with Tabu Search) approach to the VRPSD appeared in the literature. From application aspect, in addition to solve real problem in optimizing waste collection using the proposed algorithm, we also develop software package for solving VRPSD. And the result of this study will be presented and published at the international publications/ journal.

Along the recent increase in the demand for an efficient management system for VRP and logistics and the advances in computer and information technology, the importance of being able to effectively make use of the huge amount of on-line information and it has become important for a wide range of applications. Cost efficient routing of vehicles play an important role to a wide range of industries. As indicated earlier, our focus would be to work on the problem related to VRP for solid waste collection in Johor Bahru.

1.7 Thesis Proposal Layout

This thesis proposal contains eight chapters. The first chapter is the introduction. This chapter gives an introduction to the background of the problem, the statement of the problem, objectives and scope of the study, and significance of the study. Chapter two is the Literature Review. This chapter presents a literature review about the Vehicle Routing Problem with Stochastic Demands, solution techniques appeared in literature and also techniques which may be applied for solving VRPSD. Chapter three, the Research Methodology, presents the direction of the study and an overview of the methods used.

Chapter four explores the development of heuristic method based on Tabu Search for solving single VRPSD. The discussion of this chapter begins with the detail development of TS for single VRPSD, followed by the experimental results and discussion. Chapter five explores the dynamism of Genetic Algorithm in solving the VRPSD. Chapter six provides the introduction and applications of Programming with Microsoft Visual Studio C++ 6.0 using Ant Colony System (ACS) and Simulated Annealing (SA). Chapter seven discusses the comparison between ACS and SA. The last chapter, Chapter eight, is Conclusion and Recommendations for further research.

CHAPTER 2

REVIEW OF THE LITERATURE

2.1 Introduction

The process of solving Vehicle Routing Problem (VRP) takes place again and again in many business operations. This has led many researchers in this field to develop solution techniques to generate an optimal solution for solving VRP which may also be applied to other variant of VRPs. This research as indicated in the previous chapter focuses on developing solution methods for solving VRP with Stochastic Demands (VRPSD). This chapter commences with the description of the VRPSD, the preview on the solution frameworks, and mathematical model of VRPSD fully with previous works on VRPSD. These are followed by a discussion on issues related to the use of heuristic methods, the basic concept of metaheuristics and hybrid metaheuristcs relevant to this study, and some criteria for classification of metaheuristics. Finally, the last section of this chapter discusses the taxonomy of hybrid metaheuristics.

2.2 Vehicle Routing Problem with Stochastic Demands (VRPSD)

VRP and its variants are at the core of many industrial applications in transportation logistics. In this study a variant of VRP is studied, where customer demands are not deterministically known but unknown until the time when the vehicle arrives at the customer location. To deal with this problem, the VRP is extended to cover the more realistic case of uncertainty in customer demands by using VRP with Stochastic Demands model. The customer demands are unknown but assumed to follow specific probability distribution according to the past experience about customer demands.

2.2.1 Real World Applications

The VRPSD finds its applications in all those cases where it is impossible to know the demand of the customers before the vehicles arrive at customer's location. The pick up of garbage is one of them. It is of course impossible to know a priori how much garbage has to be collected at each customer location. It is possible that the total quantity of garbage could exceed the vehicle capacity. In such situation, the vehicle needs to go to the depot to replenish and then resume its route.

The delivery of petrol stations is another case subject to the stochasticity of demand. When customer issues the order, it is still unknown how much he will sell in the time between the order and the new delivery. Another related application is in the cash replenishment of automatic teller machines of a bank. The daily demand for cash at a specific machine is a random variable, while the maximum amount of cash that might be carried by a vehicle at any time is specified by a security policy. Not all machines may be able to be supplied on a route from its designated vehicle, forcing a decision to be made as to how to supply the machines when restocking demand may exceed vehicle capacity.

According to Yang et al. (2000), VRPSD also can appear in case of salespersons on peddle routes, salesperson stock their trucks with goods that they anticipate can be sold on the route. Visits are made to predetermined customer sites with the intent of replacing depleted stock. Because there is no advanced reporting of the current inventory level or the size of a replenishment order, the amount to be sold is not known until the site is visited. Examples of such peddle routes are (1) beer distribution to retail outlets, (2) supply of baked goods at food stores, (3) replenishment of liquid gas at research laboratories, and (4) stocking of vending machines.

2.2.2 Solution Frameworks for VRPSD

Dror et al. (1989) mentioned that there are two solution frameworks for Stochastic VRP (including VRPSD), namely stochastic programming and Markov decision process.

2.2.2.1 Stochastic Programming

By a mathematical programming problem, we usually understand a problem of the type

Minimize $Z_{MP} = f(x)$

subject to: $g_i(x) \leq 0$, (i = 0, 1, …, m),

$x \in S \subseteq R^n$,

$x \geq 0$

where the real functions f, g_i (i = 0, 1, …, m) are assumed to be deterministic and known.

In general, the stochastic programming is mathematical programming where one or more parameters are random variables. Such cases seem typical of real-life problems, where it may be difficult to determine the values of the parameters with certainty. The stochastic programming can be described as

Minimize $Z_{SP} = f(x, \xi)$

subject to: $g_i(x,\xi) \le 0$, (i = 0, 1, …, m)

$x \in S \subseteq R^n$,

$x \ge 0$.

where ξ is a vector of random variables defined on a probability space $(\Omega, \Sigma, \mathrm{P})$, hence ($f(x,\xi)$, $g_i(x,\xi)$, i = 0, 1, …, m) is a random vector.

Stochastic programming terminology distinguishes between "wait and see" and "here and now" situations. The SVRP more naturally belongs to the "here and now" category since usually routes have to be planned in anticipation of customer demands which become only known only in the process of executing the planned routing sequence. Again, the problems in this category are usually modeled in two ways: so-called chance constrained stochastic programming and stochastic programming with recourse. (Dror et al., 1989)

(i) Chance constrained programming

Minimize $Z_{CPP} = E_i[f(x,\xi)]$

subject to: $x \in S$

$P(\xi | g_i(x,\xi) \le 0) \ge 1-\alpha$, i = 0, 1, …, m

or $P(\xi | g_i(x,\xi) \le 0) \ge 1-\alpha_i$, i = 0, 1, …, m

The name "chance constrained" follows from the fact that each constraint is realized with a minimum probability of $1-\alpha_i$, $0 < \alpha_i < 1$ (Taha, 2003). Laporte et al. (1989) in Dror et al. (1989) define two-index variables x_{ij} instead of three-index variables x_{ijk}, but their model cannot take into account heterogeneous vehicle capacities. Let x_{ij} be a binary variable equal to 1 if and only if a vehicle travels on arc (*i*,*j*). Also let *m* be the number of vehicles available, T_m is the set of all feasible solutions to the *m*-TSP and α is the maximum allowable probability of route failure. Then the Laporte et al. model (called CCP1), adapted to the case of general distance matrices, can be written as

Minimize $Z_{CCP1} = \sum_{i,j} c_{ij} x_{ij}$

subject to $\sum_{i\in S, j\notin S} x_{ij} \geq V_\alpha(S)$, $(S \subseteq \{1,...,n\}; |S| \geq 2)$

$x = (x_{ij}) \in T_m$

Here $V_\alpha(S)$ is the minimum number of vehicles required to serve all customers of S so that the probability of route failure in S does not exceed α, i.e., $V_\alpha(S)$ is the smallest integer such that $P(\sum_{i\in S} \xi_i > QV_\alpha(S)) \leq \alpha)$. The value of $V_\alpha(S)$ is easily determined provided the probability distribution of $\sum_{i\in S} \xi_i$ is known or even approximated for any non-empty subset S of $\{1, \ldots, n\}$.

(ii) Stochastic Programming with Recourse

Minimize $Z_{SPR} = E_i[f(x,\xi) + \phi(x,\xi)]$

subject to: $x \in S$

where $\phi(x,\xi)$ is some non-negative real function $H(g_i(x,\xi))$ taking the value 0 if ($g_i(x,\xi) \leq 0$). This expresses the fact that if all the constraints are satisfied, then we are concerned with the value of the objective function *f* only, but when a constraints violation occurs, we have to add a penalty to the objective function. In the SVRP, $\phi(x,\xi)$ correspond to the cost of corrective or recourse actions. (Dror et al., 1989)

Laporte and Loveaux (1989) in Dror et al. (1989) propose a stochastic models with recourse with full service (i.e., no split deliveries) is assumed and two cases are considered:

1. The SVRP with *early information*: all customer demands are known after the route is constructed but before the vehicle leaves the depot and route breaks are planned in advance so as to minimize their expected cost;
2. The SVRP with *late information*: breaks and failures coincide.

In both cases, it is assumed that the planned routing sequence is unaltered by failures or breaks. The two models differ only in their recourse function. The general model (called SPR1) given as

Minimize $Z_{SPR1} = cx + E_i[\phi(x,\xi)]$

subject to $x \in T_m$

where $\phi(x,\xi)$ is the recourse function.

For a more detailed analysis we refer to the original paper since the explicit recourse representation is quite complicated.

2.2.2.2 A Markov Decision Process Model

Consider a SVRP with a single vehicle of fixed capacity *Q* located at the depot with the assumption that no customer demand exceeds *Q*. In addition, assume that the exact demand becomes known immediately upon arrival at the customer location (i.e., before replenishment begins). Upon the arrival at a customer location, two possible actions can be taken prior to the start of service:

(i) not to replenish the customer and then move to another location (which could be another customer or the depot),

(ii) to replenish the customer and then move to another location.

Dror et al. (1989) assumed a non preemptive service policy which implies that if customer service (replenishment) has started, it is not interrupt until either the customer is fully replenished or the vehicle is empty, in which case the only option is to return to the depot. When there are no more customers to be replenished, then again the only option is to return to the depot. This is the final state of the system. The initial state is condition that there is a full vehicle at the depot with all *n* customers to be serviced.

The system is observed each time the vehicle arrives at one of the locations of {0, 1, …, *n*}. Let $\tau_0 = 0, \tau_1, \tau_2, ...(\tau_i \leq \tau_{i+1}$; *i* = 0, 1, …) be the times (cumulative distance traveled by the vehicle) at which these events occur. Note that τ_i corresponds to the time of the i^{th} arrival which is not necessarily an arrival time at location *i*. These are called *transition times* and correspond to the times at which decisions are taken. The *state* of the system at a transition time τ_k is described by a vector $s = (r, l, x_1, ..., x_n)$, where $r \in$ {0, 1, …, *n*} denotes the position of the vehicle and $l \in$ [0,*Q*] describes the stock level in the vehicle (if r = 0 the l=*Q* i.e., the vehicle is full). If customer *i* has been visited, then the exact demand is known and x_i represents the remaining demand ($x_i \geq 0$). If customer I has not yet been visited, then the demand is unknown and is denoted by x_i = -1. The *state space* is a subset S of {0, 1, …, n} × [0, Q] × $(\{-1\} \cup [0,Q])^n$ which satisfies the above conditions.

At each transition time, a decision *d* is selected from the decision space $D = \{d_1, d_2\} \times \{0,1,...,n\}$ where $d = (d_1, i)$ means that the vehicle goes to the customer *i* from its present location, say *j*, without servicing *j* and $d = (d_2, i)$ corresponds to the case where the vehicle first replenishes *j* before going to *i*. The decision $d = (\cdot, i), (i \neq 0)$ is admissible even if *l* = 0, which represents the value of collecting the information on the demand at *i* without being able to satisfy any part of that demand. If x_i = 0 for all I, then the only admissible action is $d = (d_1, 0)$ i.e., returning to the depot. For each $s \in S$, let $D(s) \subset D$ denote the set of admissible decisions when the system is in state *s*, and $\tau = \{(s,d) | d \in D(s)\}$ the set of admissible state-decision pairs.

At transition time τ_k, the system is in some state $s_k \in S$ and a decision $d_k \in D(s_k)$ is taken. The time τ_{k+1} of the next transition is deterministic (specified by the distance matrix *C*), and the next state s_{k+1} is generated according to the probability distribution which governs the demand variables. Suppose $s_k = (r_{\tau_k}, l, x_1, ..., x_n) \in S$ is the observed state at the current transition time τ_k, and $d = (d_*, i) \in D(s)$ is the decision taken, where $d_* \in \{d_1, d_2\}$. Then the time (distance) until the next event is simply the difference $\tau_{k+1} - \tau_k = c_{r_{\tau_k}, i}$ where r_{τ_k} is the location of the vehicle at transition time τ_k (we implicitly assume that service time is zero). Let $p(\cdot | s, d)$ be the transition law, i.e., for every Borel subset $\overline{S}$ of *S*, $p(\overline{S} | s, d)$ is the probability that the next state belongs to $\overline{S}$, given *s* and *d*.

A control policy is a function μ which associates with each state $s \in S$ a decision $d = \mu(s) \in D(s)$. The aim of the decision maker is to find a policy which minimizes the expected cost of a decision sequence, starting at $s_0 = (0, Q, -1, ..., -1)$

and terminating at $s^* = (0, Q, 0, ..., 0)$. More precisely, one seeks a policy μ^* which minimizes $E_\mu[\sum_{k=0}^{T} c_{r_{\tau_k}, r_{\tau_{k+1}}}]$ where T is the random number of transitions. (Dror et al., 1989)

2.2.3 Mathematical Formulation of Single VRPSD

VRPSD is a variant of VRP with the goal is to find a vehicle route and a restocking policy at each node (a threshold) that minimizes the total expected cost. The costs under consideration are:

- Cost of traveling from one customer to another as planned.
- Restocking cost: the cost of traveling back to the depot for restocking.
- The cost of returning to depot for restocking caused by the remaining stock in the vehicle being insufficient to satisfy demand upon arrival at a customer location. This route-failure cost is a fixed nonnegative cost *b* plus a cost of traveling to the depot and back to the route.

Let 0 → 1 → 2 … *j* → *j*+1 … → *n* be a particular vehicle route. Upon the service completion at customer *j*, suppose the vehicle has a remaining load *q* (or the residual capacity of the vehicle after having serviced customer *j*), and let $f_j(q)$ denote the total expected cost from node j onward. If S_j represents the set of all possible loads that a vehicle can have after service completion at customer *j*, then, $f_j(q)$ for $q \in S_j$ satisfies

$$f_j(q) = \min imum \begin{cases} f_j^p(q) \\ f_j^r(q) \end{cases}, \tag{2.1}$$

where

$$f_j^p(q) = c_{j,j+1} + \sum_{k:\xi^k \le q} f_{j+1}(q - \xi^k) p_{j+1,k}$$

$$+ \sum_{k:\xi^k > q} [b + 2c_{j+1,0} + f_{j+1}(q + Q - \xi^k)] p_{j+1,k} \tag{2.2}$$

and

$$f_j^r(q) = c_{j,0} + c_{0,j+1} + \sum_{k=1}^{K} f_{j+1}(Q - \xi^k) p_{j+1,k} \tag{2.3}$$

with the boundary condition

$$f_n(q) = c_{n,0}, q \in S_n \tag{2.4}$$

In equations (2.2-2.4), $f_j^p(q)$ represents the expected cost of going directly to the next node, whereas $f_j^r(q)$ represents the expected cost of the restocking action. These equations are used to recursively determine the objective value of the planned vehicle route and the optimal sequence of decisions after customers are served. (Bianchi et al., 2005) In principle, this procedure leads to a dynamic programming since each time a customer demand is revealed, a decision has to be taken as to where the vehicle should proceed.

2.2.4 Threshold and Expected Cost Evaluation

The expected cost-to-go in case of restocking, is constant in *q*, since in case of restocking the vehicle will have full capacity *Q* before serving the next customer, whatever the current capacity *q* is. On the other hand, $f_j^p(q)$ is a monotonically non-increasing function in *q*, for every fixed customer *j*. Therefore there is a capacity threshold value h_j such that, if the vehicle has more than this value of residual goods, then the best policy is to proceed to the next planned customer, otherwise it is better to go back to the depot for replenish, as seen in Figure 2.1. (Yang et al., 2000)

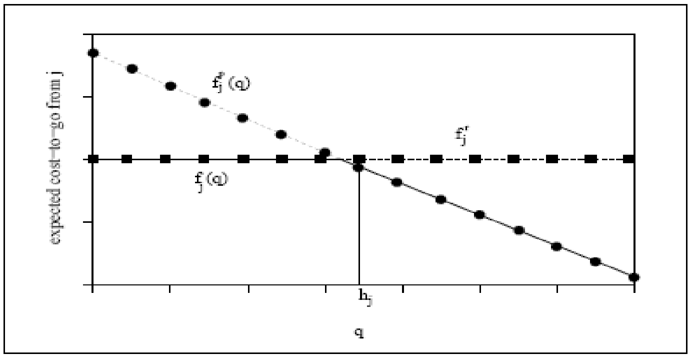
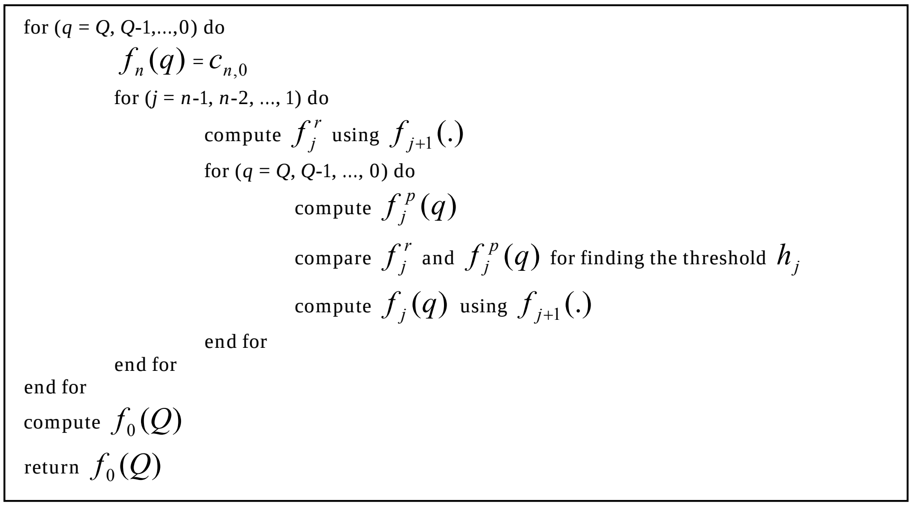


Figure 2.1. Function of *q*

The algorithm in Figure 2.2 is an implementation of the above dynamic programming recursion for the calculation of $f_0(Q)$ and of the thresholds (Bianchi et al., 2005).



```
for ($q$ = $Q$, $Q$-1,...,0) do
        $f_n(q) = c_{n,0}$
        for ($j$ = $n$-1, $n$-2, ..., 1) do
                compute $f_j^r$ using $f_{j+1}(.)$
                for ($q$ = $Q$, $Q$-1, ..., 0) do
                        compute $f_j^p(q)$
                        compare $f_j^r$ and $f_j^p(q)$ for finding the threshold $h_j$
                        compute $f_j(q)$ using $f_{j+1}(.)$
                end for
        end for
end for
compute $f_0(Q)$
return $f_0(Q)$
```

Figure 2.2. Algorithm for the computation of the VRPSD objective function $f_0(Q)$

2.2.5 Multiple homogeneous VRPSD

Given a fleet of vehicles indexed by $m \in M$, is based at a depot with each capacities Q, the problem is to find a set of a priori routes (and the associated optimal restocking policies), each to be served by a vehicle, such that customer demands on the route are fully satisfied and the total expected cost is minimized. The total demand of customers assigned to each route does not exceed the vehicle capacity of the vehicle assigned to it, and the total cost is minimized.

In deterministic vehicle routing problems, a single route will always be optimal if the vehicle has enough capacity to carry all customer demand. However, multiple vehicles (routes) are needed when the total customer demand exceeds the vehicle capacity. It is assumed that there are m vehicles for n customer nodes. Let $r = \{r_1, r_2, ..., r_m\}$ be a particular solution in which r_i is the sub tour dedicated to i^{th} vehicle. The function to be minimized is the sum of expected cost for all vehicles whereas the function of each vehicle is similar with the single vehicle. (Yang et al., 2000)

2.2.6 Previous Works on the VRPSD

The summary of main contributions to solve VRPSD and similar problems is given in Table 2.1.

Table 2.1. Previous works on VRPSD

Authors	**Methods**	**Description**
Tillman (1969)	Heuristic [multiple vehicles]	the first to propose simple heuristic algorithm for the VRPSD in the case where there were *multiple terminal deliveries*
Dror, Laporte and Trudeau (1989)	Theory	overviewed concepts and main issues in VRPSD along with some properties of optimal solutions. They also presented a new solution framework using Markovian decision processes from theoretical point.
Bertsimas, Jaillet, and Odoni (1990)	Theory [single VRPSD]	introduced the idea of a priori optimization as a strategy competitive to the strategy of re-optimization. It was shown that a priori and re-optimization strategies have on average very close behaviour.
Bertsimas (1992)	Theory [single VRPSD]	compared performance of cyclic heuristic and reoptimization. He gave analytical evidence that this approach, which is based on finding an a priori sequence, performs quite well especially if the distribution of the demand of the customers is the same.

Table 2.1. Previous works on VRPSD (continued1)

Authors	**Methods**	**Description**
Gendreau, Laporte and Seguin (1996)	Tabu Search [multiple homogeneous vehicles]	developed a tabu search algorithm called TABUSTOCH; this algorithm is to be employed when instances become too large to be solved exactly by the L-shaped method. Comparison with known optimal solutions (exact algorithm) whose sizes vary from 6 to 46 customers indicate that the heuristic produces on optimal solution in 89.45% of cases, with an average deviation of 0.38% from optimality.
Yang’s thesis (1996)	Heuristic [single and multiple homogeneous]	developed optimal restocking policy in conjunction with routing decisions for a priori VRPSD.
Yang, Mathur, and Ballou (2000)	Heuristics [single and multiple homogeneous vehicles]	investigated the route first-cluster second and the cluster first-route second. Both algorithms seem to be efficient and robust for small size instances, as shown by comparisons with branch and bound solutions to instances up to 15 customers. They also adapt the OrOpt local search to the stochastic case.
Secomandi (2001)	Rollout policy [single vehicle]	considered re-optimization-type routing policy by means of rollout policy.
Laporte, Louveaux and van Hamme (2002)	Exact algorithm [between 2 and 4 vehicles]	proposed an Integer L-Shaped algorithm for the VRPSD for instances involving between 25 and 100 vertices where demands follow a Poisson or a normal distribution. The objective is to minimize the expected solution cost.
Chepuri and Homem-de-Mello (2005)	Heuristic [single vehicle]	proposed a new heuristic method based on Cross-Entropy method for VRPSD.
Bianchi et al (2005)	Metaheuristics [single vehicle]	considered basic implementation of five metaheuristics: Iterated Local Search, Tabu Search, Simulated Annealing, ACO and Genetic Algorithm that found better solution quality in respect to cyclic heuristic.

Table 2.1. Previous works on VRPSD (continued2)

Authors	**Methods**	**Description**
Irhamah (2006)	Metaheuristics [single, homogeneous and heterogeneous vehicles]	- based on Hybrid Genetic Algorithm with Tabu Search where TS as a mutation operator - management of initial population: the inclusion of constructive and insertion heuristic and the implementation of the concept of statistical control - first considering heterogeneous VRPSD

2.3 Heuristic Methods

Silver et al. (1980) define a heuristic method as a procedure for solving well-defined mathematical problem by intuitive approach in which the structure of the problem can be interpreted and exploited intelligently to obtain a reasonable solution.

2.3.1 Why Use a Heuristic Method?

According to Silver et al. (1980), there are several possible reasons for using heuristic methods of solution. These include:

a. The mathematical problem is of such a nature that an analytic (closed form) or iterative solution procedure is unknown.

b. Although an exact analytic or iterative solution procedure may exist, it may be computationally prohibitive to use or perhaps unrealistic in its data requirements.

c. The heuristic method, by design, may be simpler for the decision maker to understand, hence markedly increasing the chances of implementation.

d. For well-defined problem that can be solved optimally a heuristic method can be used for learning purposes, e.g., to develop an intuitive feeling as to what variables are important. (This closely parallels one of the primary reasons for using simulation methods in operational research.)

e. A heuristic may be used as part of an iterative procedure that guarantees the finding of an optimal solution. Two distinct possibilities exist:
   i. To easily obtain an initial feasible solution.
   ii. To make a decision at an intermediate step of an exact solution procedure.

f. Heuristics can be used to give such 'good' starting solutions. In implicit enumeration approaches to problem solving a good starting solution can give a bound that drastically reduces the computational effort.

2.3.2 Measuring the Quality of a Heuristic

Silver et al. (1980) stated that a good heuristic should possess the following four properties:

1. Realistic computational effort to obtain solution.
2. The solution should be close to the optimum on the average, i.e., we want good performance on the average.
3. The chance of a very poor solution (i.e., far from the optimum) should be law.
4. The heuristic should be as simple as possible for the user to understand, preferably explainable in intuitive terms, particularly if it is to be used manually.

Here are several ways for measuring the quality of heuristics:

1. Comparison with the optimum solution.
2. Problem relaxation – bounding.
3. Extreme value statistical methods.
4. Other comparisons.
   i. Comparison with an enumerative method, requiring much more computational effort that is terminated after a large amount of computation, but likely without having found the optimal solution.
   ii. Comparison with performance of the decision maker, either during an earlier time frame or directly in parallel – there are compelling arguments

for this type of comparison. Identification of significant improvement over existing procedures is probably much more important for encouraging implementation than any proof of optimality, or nearness to optimality.

iii. Comparison with other heuristic procedures – where other heuristic solution methods have been proposed and/or used, one certainly can compare 'our' heuristic against the others.

iv. Comparison with a 'random' decision rule – an extreme type of heuristic is where one makes a decision completely at random.

5. Worst-case behaviour.

2.3.3 Types of heuristic methods

It should be emphasized that the categories are not meant to be mutually exclusive.

1. Decomposition methods

   Here the problem under consideration is broken into smaller parts that are solved separately, but taking account, at least in a crude way, of possible interactions among the parts.

2. Inductive methods.

   The idea here is to generalize from smaller (or somewhat simpler) versions of the same problem.

3. Feature extraction (or reduction) methods.

   The general approach here is to first obtain the optimal solutions to several numerical cases under consideration. Common features of these solutions are extracted and are assumed to hold in general.

4. Methods involving model manipulation:

   - modification of the objective function.
   - relaxation of certain constraints, some of which may be flexible in any event.
   - change nature of probability distribution.

- Aggregation of variables, the idea being to reduce the number of decision variables.

5. Constructive methods.
6. Local improvement methods.

   In contrast with the constructive procedures, local improvement starts with a feasible solution and improve upon it iteratively.

2.4 Metaheuristics

The term metaheuristic is firstly introduced by Fred Glover, derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* which means “to find”, while the suffix *meta* means “beyond, in an upper level”. Before this term was widely adopted, metaheuristics were often called modern heuristics (Reeves, 1993). Examples of metaheuristics include-but not limited to- Ant Colony Optimization, Evolutionary Computation including Genetic Algorithm, Iterated Local Search, Simulated Annealing and Tabu Search. Nowadays metaheuristics are widely used to solve important practical combinatorial optimization problems. However, due to the variety of techniques and concepts comprised by metaheuristics, there is still no commonly agreed definition for metaheuristics.

A metaheuristic is formally defined as an iterative generation process which guides a sub-ordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions (Osman and Laporte, 1996). Stutze (1999) defined metaheuristics as typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descents by allowing the local search to escape from local optima. This is achieved by either allowing new starting solutions for the local search in a more ‘intelligent’ way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decision made during the search. But, the main difference to pure random search is that in metaheuristic algorithm randomness is not used blindly but in an intelligent biased form.

The definition used in the Metaheuristics Network (2000) is as follows. A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem. Blum and Roli (2003) summarized metaheuristics as high-level strategies for exploring search spaces by using different methods. Of great importance hereby is that a dynamic balance is given between diversification and intensification. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. These terms stem from the tabu search field and it is important to clarify that the terms exploration and exploitation are sometimes used instead, for example in the Evolutionary Computation field, with a more restricted meaning. In fact the notions of exploration and exploitation often refer to rather short term strategies tied to randomness, whereas intensification and diversification also refer to medium and long term strategies based on the usage of memory. The use of the terms diversification and intensification in heir initial meaning becomes more and more accepted by the whole field of metaheuristics.

According to Glover (2003) in Raidl (2006), "… these methods have over time also come to include any procedure for problem solving that employs a strategy for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighbourhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes."

2.4.1 Classification of Metaheuristics

There are several ways to classify and describe metaheuristic algorithms. Blum and Roli (2003) summarized the most important way to classify metaheuristics as below.

- *Nature-inspired vs. non-nature inspired.*

Perhaps, the most intuitive way of classifying metaheuristics is based on the origins of the algorithm. There are nature-inspired algorithms, like Genetic Algorithms and Ant Algorithms, and non nature-inspired ones such as Tabu Search and Iterated Local Search. In our opinion this classification is not very meaningful for the following two reasons. First, many recent hybrid algorithms do not fit either class (or, in a sense, they fit both at the same time). Second, it is sometimes difficult to clearly attribute an algorithm to one of the two classes. So, for example, one might ask the question if the use of memory in Tabu Search is not nature-inspired as well.

- *Population-based vs. single point search.*

Another characteristic that can be used for the classification of metaheuristics is the number of solutions used at the same time: Does the algorithm work on a population or on a single solution at any time? Algorithms working on single solutions are called *trajectory methods* and encompass local search-based metaheuristics. They all share the property of describing a trajectory in the search space during the search process. Population-based metaheuristics, on the contrary, perform search processes which describe the evolution of a set of points in the search space.

- *Dynamic vs. static objective function.*

Metaheuristics can also be classified according to the way they make use of the objective function. While some algorithms keep the objective function given in the problem representation “as it is”, some others, like Guided Local Search (GLS), modify it during the search. The idea behind this approach is to escape from local minima by modifying the search landscape. Accordingly, during the search the objective function is altered by trying to incorporate information collected during the search process.

- *One vs. various neighborhood structures.*

Most metaheuristic algorithms work on one single neighborhood structure. In other words, the fitness landscape topology does not change in the course of the algorithm. Other metaheuristics, such as Variable Neighborhood Search (VNS), use a set of neighborhood structures which gives the possibility to diversify the search by swapping between different fitness landscapes.

- *Memory usage vs. memory-less methods.*

A very important feature to classify metaheuristics is the use they make of the search history, that is, whether they use memory or not. Memory-less algorithms perform a Markov process, as the information they exclusively use to determine the next action is the current state of the search process. There are several different ways of making use of memory. Usually we differentiate between the use of short term and long term memory. The first usually keeps track of recently performed moves, visited solutions or, in general, decisions taken. The second is usually an accumulation of synthetic parameters about the search. The use of memory is nowadays recognized as one of the fundamental elements of a powerful metaheuristic.

In the rest of this study, we use the way to classify metaheuristics according to the single point vs. population-based search classification, which divides metaheuristics into trajectory methods and population-based methods. This choice is motivated by the fact that this categorization permits a clearer description of the algorithms. Moreover, a current trend is the hybridization of methods in the direction of the integration of single point search algorithms in population-based ones.

2.4.2 Trajectory Methods

2.4.2.1 Basic Local Search: Iterative Improvement

The basic local search is usually called iterative improvement, since each move is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it finds a local minimum.

2.4.2.2 Simulated Annealing

Simulated Annealing (SA) is commonly said to be the oldest among the metaheuristics and surely one of the first algorithms that has an explicit strategy to escape from local minima. The origins of the algorithm are in statistical mechanics (Metropolis algorithm) and it was first presented as a search algorithm for CO problem in Kirkpatrick et al. (1983) (given some assumptions on the cooling schedule of the temperature, etc.) that could be shown to converge to an optimal solution. The fundamental idea is to allow moves resulting in solutions of worse quality than the current solution (uphill moves) in order to escape from local minima. The probability of doing such a move is decreased during the search.

Simulated Annealing comes from the annealing process of solids. A solid is heated until its melts, and then the temperature of the solid is slowly decreased (according to annealing schedule) until the solid reaches the lowest energy state or the ground state. If the initial temperature is decreased rapidly, the solid at the ground state will have many defects or imperfections. An easy implementation of the algorithm makes it very easy to adapt a local search method (e.g. best improvement local search) to a simulated annealing algorithm, usually rendering the local search with much better results. But although it is proven to converge to the optimum, it converges in infinite time. Not only for this reason, but also since we have to cool down slowly, the algorithm is usually not faster than its contemporaries.

2.4.2.3 Tabu Search

Tabu Search (TS) was proposed by Glover in 1986. A description of the method and its concepts can be found in Glover and Laguna (1997). The basic principle of TS is to pursue a best improvement Local Search whenever it encounters a local minimum by allowing non-improving moves, cycling back to previously visited solutions is prevented by the use of memories called tabu lists that record the recent history of the search, a key idea that can be linked to Artificial Intelligence concepts. It is also important to remark that Glover did not see TS as a proper heuristic, but rather as a metaheuristic, i.e., a general strategy for guiding and controlling “inner” heuristics specifically tailored to the problems at hand (Gendreau, 2002). Tabu Search (TS) is among the most cited and used metaheuristics for combinatorial problems. Many computational experiments have shown that TS has now become an established optimization technique which can compete with almost all known techniques and which – by its flexibility – can beat many classical procedures.

The word tabu (or taboo) comes from Tongan, a language of Polynesia, where it was used by the aborigines of Tonga Island to indicate things that can not be touched because they are sacred. According to Webster’s Dictionary, the word now also means “a prohibition imposed by social custom as a protective measure: or of something “banned as constituting a risk”. These current more pragmatic senses of the word accord well with the theme of tabu search. The risk to be avoided in this case is that of following a counter-productive course, including one which may lead to entrapment without hope of escape. On the other hand, as in the broader social context where “protective prohibitions” are capable of being superseded when the occasion demands, the “tabus” of tabu search are to be overruled when evidence of a preferred alternative becomes compelling.

The most important association with traditional usage, however, stems from the fact that tabus as normally conceived are transmitted by means of a social memory which is subject to modification over time. This creates the fundamental link to the meaning of “tabu” in TS. The forbidden elements of TS receive their status by reliance on an evolving memory, which allows this status to shift according to time and circumstance. More particularly, TS is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. The adaptive memory feature of TS allows the implementation of procedures that implement a form of sampling (Glover and Laguna, 1997).

2.4.2.4 Explorative Local Search Methods

These are the Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighbourhood Search (VNS), Guided Local Search (GLS) and Iterated Local Search (ILS).

2.4.3 Population-based Methods

Population-based methods deal in every iteration of the algorithm with a set (i.e., a population) of solutions rather than with a single solution. As they deal with a population of solutions, population-based algorithms provide a natural, intrinsic way for the exploration of the search space. Yet the final performance depends strongly on the way the population is manipulated.

2.4.3.1 Evolutionary Computation

Evolutionary Computation (EC) algorithms are inspired by nature's capability to evolve living beings well adapted to their environment. EC algorithms can be succinctly characterized as computational models of evolutionary processes. At each iteration a number of operators is applied to the individuals of the current population to generate the individuals of the population of the next generation (iteration). Usually, EC algorithms use operator called recombination or crossover to recombine two or more individuals to produce new individuals. They also use mutation or modification operators which cause a self-adaptation of individuals. The driving force in evolutionary algorithms is the selection of individuals based on their fitness (this can be the value of an objective function or the result of a simulation experiment, or some other kind of quality measure). Individual with a higher fitness have a higher probability to be chosen as members of the population of the next iteration (or as parents for the generation of new individuals). This corresponds to the principle of survival of the fittest in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave the inspiration for EC algorithms.

There has been a variety of slightly different EC algorithms proposed over the years. Basically they fall into three different categories which have been developed independently from each other. These are Evolutionary Programming (EP), Evolutionary Strategies and Genetic Algorithms. EP arose from the desire to generate machine intelligence. While EP originally was proposed to operate on discrete representations of finite state machines, most of the present variants are used for continuous optimization problems. The latter also holds for most present variants of E, whereas GAs are mainly applied to solve combinatorial optimization problems.

Genetic Algorithms (GAs) are an effective search and optimization method that simulates the process of natural selection or survival of the fittest. Holland in 1974 developed the idea and concepts behind the Genetic Algorithm and many authors have refined his initial approach (Ortiz, et al., 2004). The purpose of GAs is to provide satisfactory results for optimization problems that are hard to solve using exhaustive techniques. The researchers who used GAs to solve complex real-world problems report good results from these techniques.

Genetic Algorithms differ from traditional search and optimization algorithm in four ways:

1. GAs work with a coding of solution set, not the solution themselves.
2. GAs search from a population of solutions, not a single solution.
3. GAs use payoff information (fitness function), not derivatives or other auxiliary knowledge.
4. GAs use probabilistic transition rules, not deterministic rules. (Goldberg, 1989)

Whereas Jung (2000) notes that there are two distinguishing characteristics of the GAs that separate them from the general optimization techniques. The first is that the GAs start with an initial set of random feasible solutions, not a single solution. The GAs generate many possible solutions to a given problem and then let them compete and mate to generate improved offspring. In conventional optimization techniques, a point to point approach is applied with a stepwise procedure to get the optimum solution. According to Gen and Cheng (1997), this approach has the danger of falling in local optima. However the GAs are population to population approach, and can escape from local optima and are very effective in global search with a multi directional search. The second characteristic is that the GAs do not have any specific functional requirement for the mathematical relationship that express a given problem. Therefore the GAs can handle any kind of objective function and constraints. It is difficult to apply simple GAs to complex optimization problem. However with some modification, GAs can solve some particular problems (Jung, 2000).

There are three major advantages when applying GA to optimization problem (Gen and Cheng, 1997).

1. Due to their evolutionary nature, GAs will search for solutions without regard to the specific inner workings of the problem. GAs can handle any kind of objective functions and any kind of constraints (linear and non linear) defined on discrete, continuous or mixed search spaces.
2. The ergodicity of evolution operators makes GA very effective at performing global search (in probability). The traditional approaches perform local search by a convergent stepwise procedure, which compares the value of nearby points and moves to the relative optimal points. Global optima can be found only if the problem possesses certain convexity properties that essentially guarantee that any local optima is a global optima.
3. GA provides a great flexibility to hybridize with domain dependent heuristics to make an efficient implementation for a specific problem.

2.4.3.2 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic approach proposed in Dorigo 1992, 1996, 1999. The inspiring source of ACO is the foraging behavior of real ants. This behavior enables ants to find shortest paths between food sources and their nest. While walking from food sources and the nest and vice versa, ants deposit a substance called pheromone on the ground. When they decide about a direction to go, they choose with higher probability paths that are marked by stronger pheromone concentrations. This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.

2.5 Hybrid Metaheuristics

Over the last decade a large number of algorithms were reported that do not purely follow the concepts of one single traditional metaheuristic, but they combine various algorithmic ideas, sometimes also from outside of the traditional metaheuristics field. These approaches are commonly referred to as hybrid metaheuristics. As for metaheuristics in general, there exist various perceptions of what a hybrid metaheuristic actually is. Looking up the meaning of hybrid in the current issue (May 2006) of the Merriam Webster dictionary yields

a. something heterogeneous in origin or composition,
b. something (as a power plant, vehicle, or electronic circuit) that has two different types of components performing essentially the same function,

while the current entry in Wiktionary defines this term as

a. offspring resulting from cross-breeeding different entities, e.g. different species,
b. something of mixed origin or composition.

2.5.1 Design Issues

According to Talbi (2002), hybridization of heuristics involves a few major issues which may be classified as design and implementation. The former category concerns the hybrid algorithm itself, involving issues such as functionality and architecture of the algorithm. The implementation consideration includes the hardware platform, programming model and environment on which the algorithm is to run.

2.5.1.1 Hierarchical classification

The structure of the hierarchical portion of the taxonomy is shown in Figure 2.3.

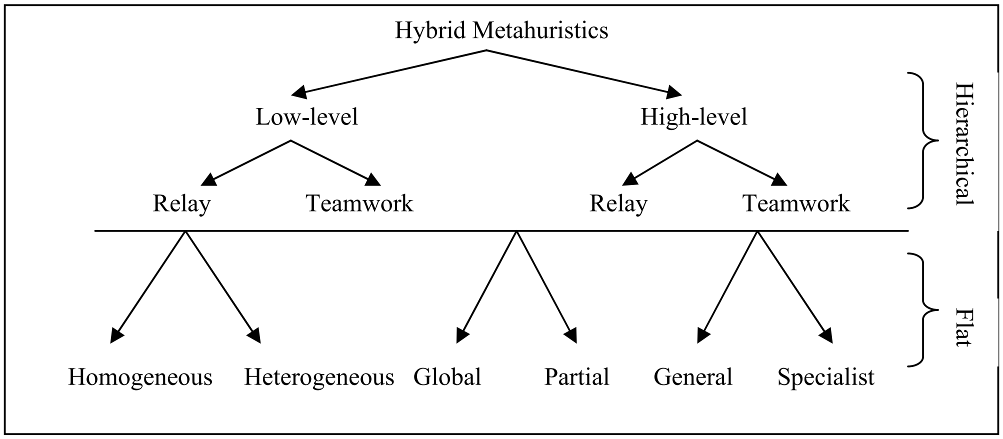


Figure 2.3. Classification of hybrid metaheuristics (design issues)

A discussion about the hierarchical portion then follows.

(i). Low-level versus high-level

The low-level hybridization addresses the functional composition of a single optimization method. In this hybrid class, a given function of a metaheuristic is replaced by another metaheuristic. In high-level hybrid algorithm, the different metaheuristics are self contained.

(ii). Relay versus teamwork

In relay hybridization, a set of metaheuristics is applied one after another, each using the output of the previous as its input, acting in a pipeline fashion. Teamwork hybridization represents cooperative optimization models, in which we have many parallel cooperating agents, where each agent carries out a search in a solution space. Four classes are derived from the following:

1. LRH (Low-level relay hybrid). This class of hybrids represent algorithms in which a given metaheuristic is embedded into a single-solution metaheuristic.
2. LTH (Low-level teamwork hybrid). Two competing goals govern the design of a metaheuristic: exploration and exploitation. Exploration is needed to ensure that every part of the space is searched enough to provide a reliable estimate of the

global optimum. Exploitation is important since the refinement of the current solution will often produce a better solution. Population-based heuristics (genetic algorithm, scatter search, ant colony, etc.) are powerful in the exploration of the search space, and weak in the exploitation of the solutions found. Therefore, most efficient population-based heuristics have been coupled with local search heuristics such as hill-climbing, simulated annealing and tabu search, which are powerful optimization methods in terms of exploitation. The two classes of algorithms have complementary strengths and weaknesses. In LTH hybrid, a metaheuristic is embedded into a population-based metaheuristic.

3. HRH (High-level relay hybrid). In HRH hybrid, self-contained metaheuristics are executed in a sequence.
4. HTH (High-level teamwork hybrid). The HTH scheme involves several self-contained algorithms performing a search in parallel, and cooperating to find an optimum. Intuitively, HTH will ultimately perform at least as well as one algorithm alone, more often perform better, each algorithm providing information to the others to help them.

2.5.1.2 Flat classification

(i) Homogeneous versus heterogeneous

In homogeneous hybrids, all the combined algorithms use the same metaheuristic. In general, different parameters are used for the algorithms. In heterogeneous algorithms, different metaheuristics are used.

(ii) Global versus partial

In global hybrids, all the algorithms search in the whole research space. The goal is to explore the search more thoroughly. All the above mentioned hybrids are global hybrids, in the sense that all the algorithms solve the whole optimization problem. In partial hybrids, the problem to be solved is decomposed into sub-problems, each one having its own search space. Then, each algorithm is dedicated to the search in one of these sub-spaces. Generally speaking, the sub-problems are all linked with each others, thus involving constraints between optima found by each algorithm. Hence, the algorithms communicate in order to respect these constraints and build a global viable solution to the problem.

(iii) Specialist versus general

All the above mentioned hybrids are general hybrids, in the sense that all the algorithms solve the same target optimization problem. Specialist hybrids combine algorithms which solve different problems. Another approach is to use a heuristic to optimize another heuristic, i.e. find the optimal values of the parameters of the heuristic.

2.5.2 Implementation Issues

The structure of the taxonomy concerning implementation issues is shown in Figure 2.4. A discussion about this taxonomy then follows.

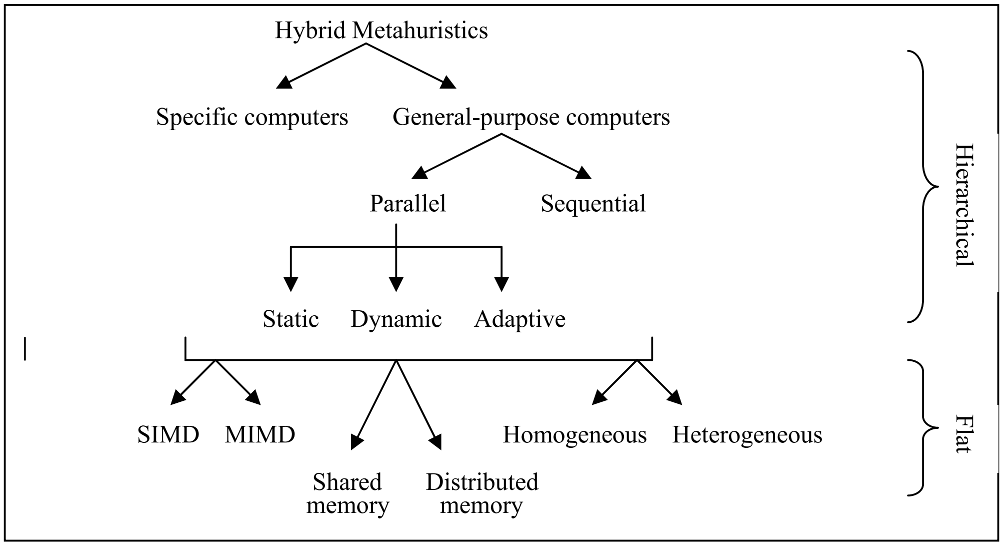


Figure 2.4. Classification of hybrid metaheuristics (implementation issues)

- **Specific versus general-purpose computers**

Application specific computers differ from general purpose ones in that usually only solve a small range of problems, but often at much higher rates and lower cost. Their internal structure is tailored for a particular problem, and thus can achieve much higher efficiency and hardware utilization than a processor which must handle a wide range of tasks.

- **Sequential versus parallel**

Most of the proposed hybrid metaheuristics are sequential programs. According to the size of problems, parallel implementations of hybrid algorithms have been considered. The easiness to use a parallel and distributed architecture has been acknowledged for the HTH hybrid model.

- **Static, dynamic or adaptive**

Parallel heuristics fall into three categories depending on whether the number and/or the location of work (tasks, data) depend or not on the load state of the target parallel machine:

- Static: this category represents parallel heuristics in which both the number of tasks of the application and the location of work (tasks or data) are generated at compilation time (static scheduling). The allocation of processors to tasks (or data) remains unchanged during the execution of the application regardless of the current state of the parallel machine. Most of the proposed parallel heuristics belong to this class.
- Dynamic: this class represents heuristics for which the number of tasks is fixed at compilation time, but the location of work (tasks, data) is determined and/or changed at run-time.
- Adaptive: parallel adaptive programs are parallel computations with a dynamically changing set of tasks. Tasks may be created or killed as a function of the load state of the parallel machine. A task is created automatically when a node becomes idle. When a node becomes busy, the task is killed. (Talbi, 2002)

2.6 Critical Reviews

As compared to the development of research in deterministic case, research on Stochastic Vehicle Routing Problem is rather undeveloped. So, the study of SVRPs is a relatively new and fast growing research area which should gain in importance with the spread of real time vehicle routing problems. The Vehicle Routing Problem with Stochastic Demands (VRPSD) is without any doubt the most studied among all SVRPs. Many algorithms for solving VRPSD have been proposed e.g. exact algorithm, heuristic, rollout policy and metaheuristics for single vehicle and multiple homogeneous vehicles.

As we know, Genetic Algorithm (GA) has been proven to be effective and successful in a wide variety of search domain and have been widely applied in solving combinatorial optimization problems, including certain types of VRP, especially where time windows are included and in TSP. Tabu Search (TS) also was known as a robust algorithm and often outperforms other heuristics in terms of computational speed and solution quality. But, the use of GA and TS in VRPSD is lacking in the literature.

Bianchi et al. (2005) have developed the basic implementation of five metaheuristics: Iterated Local Search (ILS), Tabu Search, Simulated Annealing, Ant Colony Optimization and Genetic Algorithm for single VRPSD that found better solution quality in respect to cyclic heuristic. For GA application, a small sample size of 10 individuals is used. Initial solutions are built with the randomized farthest insertion heuristic. Local search is then applied to each member of the initial population. In the steady state evolution process only one couple of parents reproduces at each generation. Tournament selection of size 2 is used to select which parents are going to be given the chance to reproduce. The crossover used in the final implementation is the Edge Recombination (ER) readapted to always start at the depot. OX and PMX were also used but ER seems to work better. It tries to build an offspring exclusively from the edges present in both parents, as outlined in Figure 2.3. Swap based mutation was used with swaps two adjacent alleles and never the depot. It is applied with an adaptive mutation rate with a maximum probability of 0.5. Order based mutation that picks 2 loci at random and exchanges their alleles, and inversion were also tried. Local search is again applied at each generation to improve the quality of the offspring. The improved offspring is then replaces the worst member of the population.

```
1.  Create an edge list from both parents, that provides for each customer all the
    other customers connected to it in at least one of the parent,
2.  start from the depot as current customer,
3.  select the customer in the edge list of the current customer, with the smallest
    number of customers left in its edge list,
4.  if there is no customer left in the edge list of the current customer then
5.      select a non yet visited customer,
6.  end if
7.  the selected customer becomes the current customer,
8.  update the edge list by removing the current customer from each adjacent
    customer list,
9.  if the tour is complete then
10.     stop.
11. else
12.     go to step 3.
13. end if.
```

Figure 2.5. Genetic Algorithm for the VRPSD

For TS implementation, Bianchi et al. uses the randomized farthest insertion heuristic as the initial solution. The neighbourhood is derived from Or-Opt local search as states in Yang et al. (2000). The strategy of selection of new solutions is a descent-ascent method with the use of tabu list.

Their experimental results show that Iterated Local Search is the only which is always among the best. At the opposite extreme is the SA, which is always among the worst. The performance of the other metaheuristics seems to be not significantly different from one to another. It has been hypothesized that this is due to the fact that all algorithms find solutions which are very near to the optimal, and thus solution values found are not very different from each other.

Gendreau et al. (1996) have developed TS for a stochastic VRP that combines two types of uncertainty: stochastic demand and stochastic customers (customers present at locations with some probabilities). They were the first to implement exact algorithm for VRPSDC in order to find optimal solutions. These can be used to assess the quality of solutions produced by another heuristics. The problem consists of designing a first stage solution that minimizes the expected cost of the second stage solution, i.e., the cost of the first stage solution plus the expected cost of recourse action. The neighbourhood of current solution is defined by removing in turn one of *q* randomly selected customers and inserting each of them either immediately before, or immediately after one of its *p* nearest neighbours. If a vertex is moved at iteration *v*, its reinsertion and displacement is tabu until iteration $v+\theta$. TABUSTOCH algorithm proceeds as follows:

1. Initialization.
2. Neighbourhood Search.
3. Incumbent update.
4. Coefficient update: update the list of tabu move.
5. Intensification or terminate.

Results indicate that the running time of the proposed heuristic is reasonable, and optimal solutions were obtained in over 89% of all instances for which an optimum was known. The success of TS seems largely due to the neighbourhood structure, to the fact that intermediate infeasible solutions are allowed, and to the proxy function used to evaluate candidate moves. This device is responsible for large time savings as *t* avoids repeated costly computations of the objective function. (Gendreau et al.,1996)

GA application in Bianchi et al. (2005) works seems to show redundancies in the use of local search for every chromosome generated twice, the first is for initial solution and the second time is for the final solution. In addition, the fact that that GA and TS seems to have the same result but can not compete the ILS also have opened a new direction in developing TS and less redundant GA for solving VRPSD. Further, it is well known that a standard Genetic Algorithm must be hybridized with another search procedure to be able to compete with other metaheuristics. So this study proposes the first implementation of hybrid GA with Tabu Search for the VRPSD that combines the advantage of GA and the strength of TS.

2.7 Summary

This chapter provides an in-depth understanding of the area of VRPSD and solution techniques for it. It began by looking at the description of VRPSD, some related works from researchers and heuristics (or metaheuristics) which can be used to solve the VRPSD. In the area of improving the solution quality, Genetic algorithm, Tabu Search and hybrid of these two algorithms are considered.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents the direction of the study and an overview of the methods used. It begins with the general steps of research framework. A description of the data source for this study and test related to it are also presented include case-study of a real-life application at solid waste collection company (Perniagaan Zawiyah Sdn. Bhd). It follows with the description of algorithm implemented with emphasis on Genetic Algorithm and Tabu Search. We provide a brief introduction to some of the terminologies used when dealing with different types of applications within a stochastic environment.

3.2 Research Framework

The general steps of research framework are summarized in Figure 3.1 while the detail will be given in the next sections.

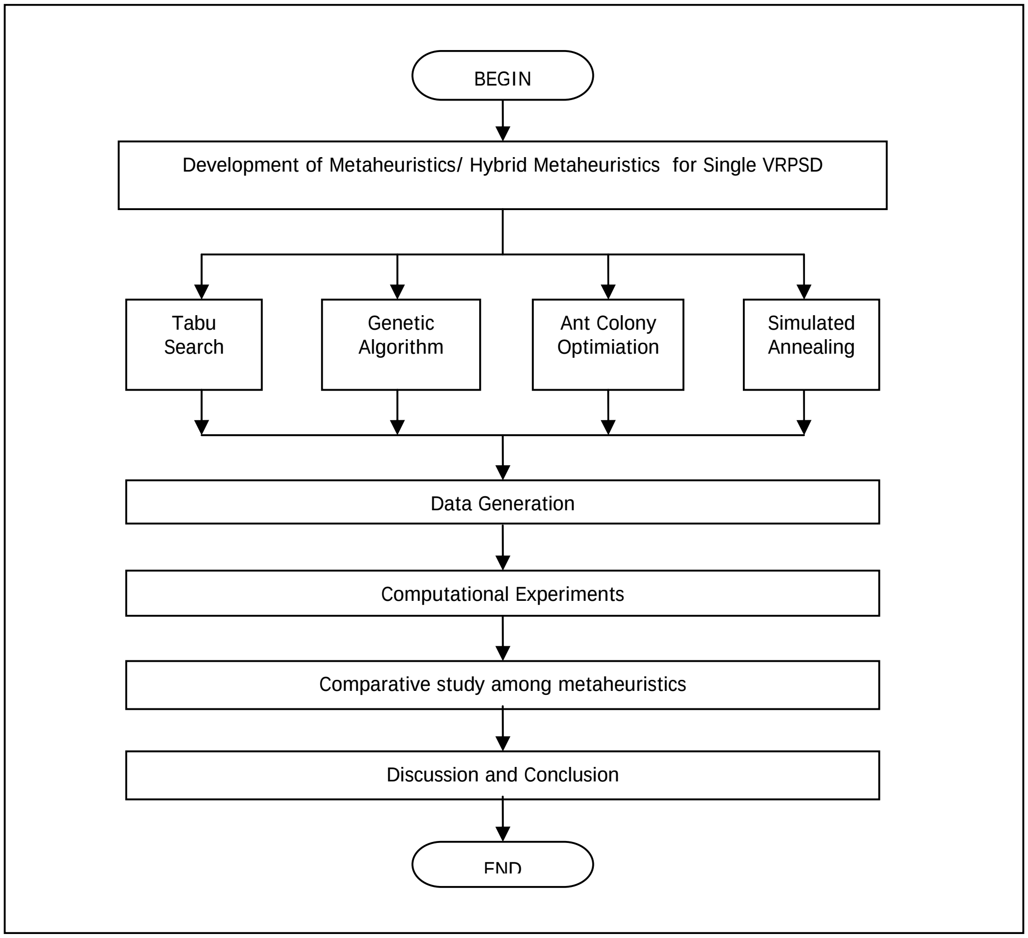

BEGIN
Development of Metaheuristics/ Hybrid Metaheuristics for Single VRPSD
Tabu Search
Genetic Algorithm
Ant Colony Optimiation
Simulated Annealing
Data Generation
Computational Experiments
Comparative study among metaheuristics
Discussion and Conclusion
END

Figure 3.1. Research Framework

3.3 Data Source

There are two kinds of data that will be used to assess the performance of the proposed algorithm:

a. Primary Data

The primary data is a real life instance of VRPSD originating from the municipal solid waste collection. The data was collected from Perniagaan Zawiyah, Sdn. Bhd., Johor Bahru, one of sub contractor of Southern Waste Management Sdn. Bhd. (SWM) – Johor Bahru branch office. SWM is a provider of a diverse range of waste management services for municipal authorities, commercial and industrial sectors for the Southern Region of Malaysia. We confine our study on residential area in Johor Bahru under municipal authorities of Majlis Perbandaran Johor Bahru.

On any given day, the company faces the problem of collecting waste from a set of customer location where the amount of waste disposal is a random variable, while each collecting truck has a limited capacity. The problem is to design a set of solid waste collection routes, each to be served by a truck such that the waste at each customer is fully collected and the total expected cost is minimized. Detail of company profile and data collected are given in Chapter 8. The variables to be measured in the data collection include:

1. Distance or coordinates of customers locations.
2. Customer demands.

b. Simulation Data

Several sets of instances will be generated to simulate real life instance of VRPSD in solid waste collection as described in Section 3.5.

3.4 Data Testing

After the data is collected, hypothesis test for distributional adequacy of demand data (whether a sample of data comes from specific distribution) will be conducted. There are two kinds of test:

1. Chi-Square Goodness of fit Test.
2. Kolmogorov Smirnov.

3.4.1 The Chi-Square Goodness of Fit Test

The chi-square goodness-of-fit test can be applied to discrete distributions. The chi-square goodness-of-fit test is applied to binned data (i.e., data put into classes). This is actually not a restriction since for non-binned data we can simply calculate a histogram or frequency table before generating the chi-square test. However, the value of the chi-square test statistic is dependent on how the data is binned. There is no optimal choice for the bin width (since the optimal bin width depends on the distribution) and most reasonable choices should produce similar, but not identical results. For the chi-square approximation it requires a sufficient sample size in order to be valid, the expected frequency should be at least 5. This test is not valid for small samples, and if some of the counts are less than five, we may need to combine some bins in the tails.

The chi-square test is defined for the hypothesis:

H_0 : The data follow a specified distribution.

H_1 : The data do not follow the specified distribution.

Test Statistic:

For the chi-square goodness-of-fit computation, the data are divided into *k* bins and the test statistic is defined as

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed frequency for bin *i* and E_i is the expected frequency for bin *i*. The expected frequency is calculated by

$$E_i = N(F(Y_u) - F(Y_l))$$

where F is the cumulative distribution function for the distribution being tested, Y_u is the upper limit for class *i*, Y_l is the lower limit for class *i*, and *N* is the sample size.

Critical Region:

Reject H_0 if $\chi^2 > \chi^2_{(\alpha,k-c)}$ where $\chi^2_{(\alpha,k-c)}$ is the chi-square percent point function with *k* - *c* degrees of freedom and a significance level of α. *k* is the number of non-empty cells and *c* = the number of estimated parameters (including location and scale parameters and shape parameters) for the distribution + 1. For example, for a 3-parameter Weibull distribution, *c* = 4. χ_α is the upper critical value from the chi-square distribution and $\chi_{1-\alpha}$ is the lower critical value from the chi-square distribution.

3.4.2 The Kolmogorov Smirnov Test

According to Randles and Wolfe (1979), the Kolmogorov-Smirnov (K-S) test is based on the empirical distribution function (ECDF). The data consist of N independent random observations, which are ordinal or interval data. Given *N ordered* data points $Y_1, Y_2, ..., Y_N$ (the Y_i are ordered from smallest to largest value), the ECDF is defined as

$F_N(Y_i)$ = proportion of sample observations less than or equal to Y_i

= number of observations less than or equal to $Y_i \ / N$

$= \dfrac{n(i)}{N}$

This is a step function that increases by $1/N$ at the value of each ordered data point.

The K-S only applies to continuous distributions and it tends to be more sensitive near the center of the distribution than at the tails. Due to these limitations, many analysts prefer to use the Anderson Darling goodness-of-fit test. However, the Anderson-Darling test is only available for a few specific distributions.

The Kolmogorov-Smirnov test is defined by:

H_0 : The data follow a specified distribution

H_1 : The data do not follow the specified distribution

Test statistic:

The Kolmogorov-Smirnov test statistic is defined as

$$D = \sup \underset{x}{remum} \ \left| F_n(Y_i) - F_0(Y_i) \right|$$

where $F_0(Y_i)$ is the theoretical cumulative distribution of the distribution being tested which must be a continuous distribution (i.e., no discrete distributions such as the binomial or Poisson), and it must be fully specified (i.e., the location, scale, and shape parameters cannot be estimated from the data).

Critical Region:

Reject H_0 if the test statistic, *D*, is greater than the critical value obtained from a table ($D \geq d(n,\alpha)$), where $d(n,\alpha)$ is the upper 100th percentile for the H_0 distribution of *D*.

3.5 Data Generation

From our literature review, there is no commonly used benchmark for the VRPSD, therefore we will generate our own test bed. We consider several sets of randomly generated instances that simulate real problem data from case study of solid waste collection.

Based on experiments reported in Gendreau et al. (1995), three factors seem to impact the difficulty of a given VRP instances: number of customers *n*, number of vehicles *m*, and filling coefficient *f*. In a stochastic environment, the filling coefficient can be defined as

$$f = \sum_{i=1}^{n} \frac{E(\xi_i)}{mQ} \tag{3.1}$$

where $E(\xi_i)$ is the expected demand of customer *i* and *Q* denotes the vehicle capacity. This is the measure of the total amount of expected demand relative to vehicle capacity and can be approximately interpreted as the expected number of loads per vehicle needed to serve all customers. In this experiment, the value of *f* is set to 0.9, as considered by Laporte et al. (2002).

Customer locations will be generated in the [100, 100] square following a discrete uniform distribution with the depot fixed at coordinate (50, 50). Each c_{ij} is then defined as travel cost from *i* to *j*, as a function of distance traveled. Without loss of generality, it is assumed that the cost of travel is RM 1 per unit distance. The distance between customers location *i* and *j* is calculated using Euclidean distance as follows:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

and it is assumed further that the distance is symmetric, that is $d_{ij} = d_{ji}$ and $d_{ii} = 0$.

We consider discrete probability distribution. The problem data will be generated using Minitab 14 software package.

3.5.1 Single vehicle

For single vehicle, the customers demands are following discrete uniform distributions: U(1,5), U(6,10) and U(11,15) respectively. There are four instances that will be generated: small instances characterized by a number of customers in the set of {10, 15} customers and larger instances in the set of {20, 50} customers. When m = 1 then the filling coefficient can be defined as

$$f = \sum_{i=1}^{n} \frac{E(\xi_i)}{Q}$$

It follows that for all instances, the a priori expected demand of each customer, denoted by $\overline{\xi}$, is 8 and $f = 8n / Q$. The values of Q for all possible setting parameters are computed by rounding $8n / f$ to the nearest integer. Each possible combination of f and n will be generated 10 times, so the reported computational results will be the average values over the number of successful problems.

3.5.2 Multiple homogeneous vehicles

The second experiment consider multiple homogeneous instances by employing filling coefficient in equation (3.1) and keeping the same filling coefficient as in the single vehicle instance case. In this experiment, we will generate three problems: 50 customers (5-6 vehicles), 75 customers (9 or 10 vehicles) and 100 customers (11 or 12 vehicles).

3.5.3 Multiple heterogeneous vehicles

Golden et al. (1984) developed test problems for the vehicle fleet size and mix routing problem which can be viewed as a special case of the heterogeneous VRP where the travel costs are the same for all vehicle types and the number of vehicles of each type is unlimited. Taillard (1999) has replaced the fixed costs by variable costs in the problem instances of Golden et al. (1984) and has specified the number of vehicles of each type available. More precisely, the travel cost c_{ij} between customers *i* and *j* is replaced by $c_{ijk} = \alpha_k c_{ij}$ when the travel is performed by a vehicle of type *k*. The variable costs have been chosen in such a way that no single vehicle type is much better or worse than any of the others. In this research, we adapted Taillard problems with a modification to the stochastic demands environment. The data for Taillard problem number 15, 17 and 20 can be seen in Table 3.1 where n is the number of customers *n* and for each vehicle type *k*: the capacity Q_k, the fixed cost f_k, the variable cost α_k and the number n_k of vehicles available.

Table 3.1. Data for heterogeneous vehicles instances

No	*n*	Vehicle type															
		A				B				C				D			
		Q	*f*	α	*n*	*Q*	*f*	α	*n*	*Q*	*f*	α	*n*	*Q*	*f*	α	*n*
15	50	50	100	1	4	100	250	1.6	3	160	450	2	2				
17	75	50	25	1	4	120	80	1.2	4	200	150	1.5	2	350	320	1.8	1
20	100	60	100	1	6	140	300	1.7	4	200	500	2	3				

The filling coefficient (as defined in equation (3.1)) used in heterogeneous vehicle takes the same value as single and multiple homogeneous vehicles. This filling coefficient and the total vehicle capacity are then used to compute the value of expected demand denoted by

$$E(\xi_i) = \frac{f.\sum n_k Q_k}{n} \tag{3.2}$$

Similar to the case of single vehicle and multiple homogeneous vehicles, customers demands are divided into three categories with expected demand of each customer is computed in (3.2). For example in Problem number 15, the total vehicle capacity is

$$\sum_{k=1}^{3} n_k Q_k = n_A.Q_A + n_B.Q_B + n_C.Q_C$$

= (4)(50)+(3)(100)+(2)(160)

= 820

It follows that the expected demand of each customer is (0.9)(820)/50 = 14.76, rounded to the nearest integer, 15. For this problem instances, customers demands are following discrete uniform distributions: U(5,15), U(10,20) and U(15,25) respectively. 10 instances will be generated for each problem number.

3.6 Genetic Algorithm

Solutions to genetic algorithm are represented by data structures, often a fixed length vector, called chromosomes. Chromosomes are composed of genes which have value called alleles, and the position of the gene in the chromosome is called locus (Goldberg, 1989). Chromosomes have been represented by vectors, strings, arrays, and tables. Alleles have been represented by a simple binary system of zeroes and ones, integers, real number, and letter of alphabet, or other symbols (Gen and Cheng, 1997). Each chromosome has a fitness value which is indicative of the suitability of the chromosome as a solution to the problem (Goldberg, 1989).

A simple genetic algorithm can be summarized as follows.

1. *Representation:* Encode the characteristics of each individual in the initial population as a chromosome (typically, a chromosome is a bit string). Set the current population to this initial population.
2. *Reproduction/ Selection:* Select two parent chromosomes from the current population and a chromosome with high fitness is more likely to be selected.
3. *Recombination:* Generate two offspring from the two parents by exchanging sub strings between parent chromosomes (crossover).
4. *Mutation:* this is a random change of a bit position in offspring chromosome.
5. Repeat steps (2), (3) and (4), until the number of chromosomes in the new population is the same as in the old population.
6. Set the current population to the new population of chromosomes.

This procedure is repeated for a fixed number of generations, or until convergence to a population of similar individuals is obtained. Then, the best chromosome generated during the search is decoded into the corresponding individual (Potvin and Bengio, 1996). The procedure of GA can be described in Figure 3.2 (Gen and Cheng, 1997).

```
Procedure: Genetic Algorithms
begin
        $t \leftarrow 0$
        initialize $P(t)$;
        evaluate $P(t)$;
        while (not termination condition) do
                recombine $P(t)$ to yield $C(t)$;
                evaluate $C(t)$;
                select $P(t + 1)$ from $P(t)$ and $C(t)$;
                $t \leftarrow t + 1$;
        end
end
```

Figure 3.2. Procedure of Genetic Algorithm

3.6.1 Selection Methods

In selection process, chromosomes are selected from the population to be parents for crossover. According to Darwin's theory of evolution the best ones survive to create new offspring. There are many methods in selecting the best chromosomes. Some of them will be described in following (Obitko, 1998).

a. Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a roulette wheel where all the chromosomes in the population are placed. The size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome: the bigger the value is, the larger the section is. For an example, see Figure 3.3 below.

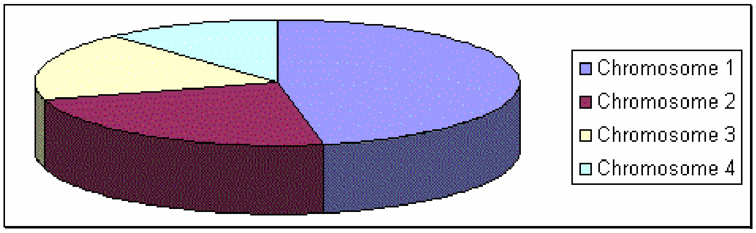


Figure 3.3. Illustration of Roulette Wheel Selection

A marble is thrown in the roulette wheel and the chromosome where it stops is selected. Clearly, the chromosomes with bigger fitness value will be selected more times. This process can be described by the following algorithm.

1. [Sum] Calculate the sum of all chromosome fitnesses in population - sum S.
2. [Select] Generate random number from the interval (θ,S) - r.
3. [Loop] Go through the population and sum the fitnesses from θ - sum s. When the sum s is greater then r, stop and return the chromosome where you are.

Of course, the step 1 is performed only once for each population.

b. Rank Selection

The previous type of selection will have problems when they are big differences between the fitness values. For example, if the best chromosome fitness is 90% of the sum of all fitnesses then the other chromosomes will have very few chances to be selected. Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2 etc. and the best will have fitness N (number of chromosomes in population). The following picture describes how the situation changes after changing fitness to the numbers determined by the ranking.

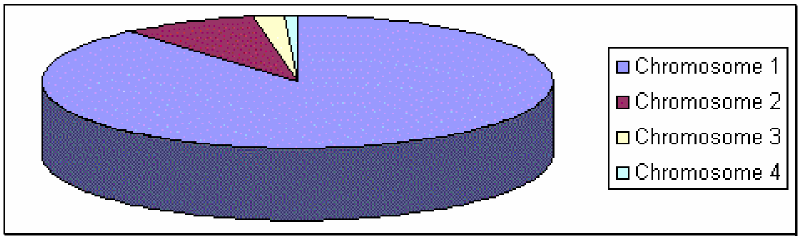


Figure 3.4. Situation before ranking (graph of fitnesses)

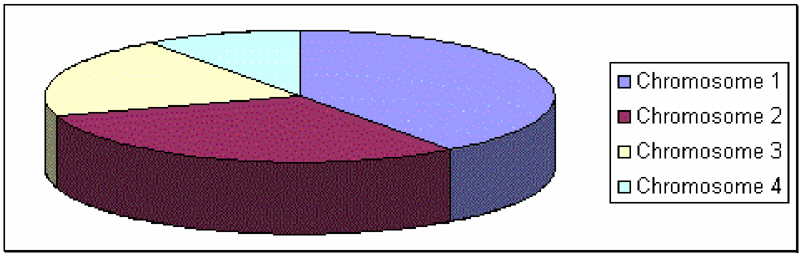


Figure 3.5. Situation after ranking (graph of order numbers)

Now all the chromosomes have a chance to be selected. However this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

c. Steady-State Selection

This is not a particular method of selecting parents. The main idea of this type of selecting to the new population is that a big part of chromosomes can survive to next generation. The steady-state selection GA works in the following way. In every generation a few good (with higher fitness) chromosomes are selected for creating new offspring. Then some bad (with lower fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

d. Elitism

When creating a new population by crossover and mutation, we have a big chance, that we will loose the best chromosome. Elitism is the name of the method that first copies the best chromosome (or few best chromosomes) to the new population. The rest of the population is constructed in ways described above. Elitism can rapidly increase the performance of GA, because it prevents a loss of the best found solution (Obitko, 1998).

3.6.2 Recombination

The simplest type of recombination is one point crossover. As illustrated in Loft and Snow (2006) where chromosomes are strings of symbols (here, 0's and 1's), there are two chromosomes; one from Parent A and one from Parent B, both the same length. A crossover point is selected randomly. The two chromosomes are cut at this point, and a new chromosome is formed by using the chromosome from Parent A before the crossover point and from Parent B after the crossover point. This is depicted in Figure 3.6.

Parent A:	0	1	1		0	1	0	1	1
Parent B:	1	0	1		1	1	0	0	0
Offspring	0	1	1		1	1	0	0	0

initial segment of A

final segment of B

Figure 3.6. One point crossover

A first generalization of one point crossover is two point crossover. In two point crossover, two crossover points are randomly chosen. Genetic material is copied from A before the first crossover point. Between the crossover points, material is taken from Parent B. After the second crossover point, material is again taken from A. This is depicted in Figure 3.7.

Parent A:	0	1		1	0	1		0	1	1
Parent B:	1	0		1	1	1		0	0	0
Offspring	0	1		1	1	1		0	1	1

initial and final segments of A

middle segment of B

Figure 3.7. Two point crossover

From two point crossover, one can imagine three point crossover, four point, five point, and so on. The logical conclusion of this is uniform crossover. In uniform crossover, each symbol in the offspring's chromosome is chosen randomly to be equal to the corresponding symbol of the chromosome of either Parent A or Parent B as shown in Figure 3.8.

Parent A:	0	1	1	0	1	0	1	1
Parent A:	1	0	1	1	1	0	0	0
Offspring	0	0	1	0	1	0	0	1

(the underlined symbols are the ones chosen for the offspring)

Figure 3.8. Uniform crossover

Loft and Snow (2006) observed that if a pattern appears in the chromosomes of both parents, then recombination will preserve that pattern. For example, both parents above have 1's as the third and fifth symbol in their chromosomes and 0 as the sixth then in one point, two point, and uniform crossover the offspring has the same pattern. Other crossovers are Partially Match Crossover (PMX), Cycle Crossover (CX), Order Crossover (OX), Matrix Crossover (MX) and Modified Order Crossover (MOX) (Bryant, 2000).

3.6.3 Mutation

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the Genetic Algorithm may be able to arrive at a better solution than what was previously possible. Mutation is an important part of the genetic search as helps to prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability rate. This probability rate should usually be set low. If it is set too high, the search will turn into a primitive random search (Mitchell, 1996).

The primary purpose of mutation is to increase variation into a population. Mutation is the most important in populations where the initial population may be a small subset of all possible solutions. It is possible, for example, that every instance of an essential bit might be zero in the initial population. In such a case, crossover could never set that bit to one; mutation, however, can set that bit to one. Mutation takes place after crossover has performed. Mutation changes randomly the new offspring. For binary encoding it is done by switching a few randomly chosen bits from 1 to 0 or from 0 to 1 (Mitchell, 1996).

A variety of mutation methods are used in GAs, e.g. inversion, insertion, displacement and reciprocal exchange mutation. Inversion mutation selects two positions within a chromosome at random and then inverts the substring between these two positions. Insertion is where individual is randomly selected and inserted in a random position. Displacement is where a subtour is selected at random and inserted in a random position. Whereas reciprocal exchange where we select two positions at random and swap them (Gen and Cheng, 1997).

Another mutation operators are 2-opt, 3-opt and Or-opt. In the 2-opt operator, two edges (a, b) and (c, d) are randomly selected from the tour and check if we can connect these four nodes in a different manner that will give a lower cost. To do this check if cab + ccd > cac + cdb. If this is the case, replace the edges (a, b) and (c, d) with the edges (a, c) and (d, b). Note that it is assumed that a, b, c and d appear in that specific order in the tour even if b and c are not connected. A 3-opt operator looks at three random edges instead of two. If there are edges (a, b), (c, d) and (e, f), check if cab + ccd + cef > cac + cbe + cdf. If it is, replace (a, b), (c, d) and (e, f) with the edges (a, c), (b, e) and (d, f).

The Or-opt operator is similar to the 3-opt. A set of connected nodes are randomly chosen and check if this string can be inserted between two other connected nodes to give a reduced cost. We can calculate this by finding the total cost of the edges being inserted and the total cost of the edges being removed. If the cost of the edges being removed is greater than the cost of those being inserted the switch is made (Bryant, 2000).

3.7 Tabu Search

TS is founded on three primary themes:

1. The use of flexible attribute-based memory structures designed to permit evaluation criteria and historical search information to be exploited more thoroughly than by rigid memory structures (as in branch and bound) or by memoryless systems (as in SA and other randomized approach).
2. An associated mechanism of control – for employing the memory structures – based on the interplay between conditions that constrain and free the search process (embodied in tabu restriction and aspiration criteria).
3. The incorporation of memory functions of different time spans, from short term to long term, to implement strategies for intensifying and diversifying the

search. (Intensification strategies reinforce move combinations and solution features historically found good, while diversification strategies drive the search into new regions). (Glover, 1990)

Some terminology in Tabu Search:

i. A move : a transition from a solution *s* to another solution *s*' in $\mathcal{N}(s)$

ii. An attribute : the elements that constitute the move

iii. Tabu list : a list of moves that are currently tabu

iv. Tabu list size : the number of iterations for which a recently accepted move is not allowed to be reversed

v. Tabu tenure : an integer number telling for how long a given move will remain tabu

The TS features are presented below:

1. *Neighbourhood structures*. The search starts with initial solution and defines a subset, $\mathcal{N}(s)$, of possible solutions that are neighbours of *s* (solution) under $\mathcal{N}$. Each neighbour is evaluated against the objective function.
2. *Short term memory*. The core of TS is embedded in its short term memory process (Glover, 1990). Tabu is used to prevent cycling when moving away from local optima through non-improving moves, it is most common to prohibit reverse moves for certain iterations. Tabu is also useful to help the search move away from previously visited portions of the search space and thus perform more extensive exploration. (Cordeau and Laporte, 2002) The neighbourhood of the current solution is restricted to the solutions that do not belong to the tabu list. At each iteration the best solution from the allowed set is chosen as the new current solution. Additionally, this solution is added to the tabu list and one of the solutions that were already in tabu list is removed (usually in FIFO order). The algorithm stops when a terminate condition is met. It might also terminate if the allowed set is empty, that is, if all the solution in $\mathcal{N}(s)$ are forbidden by the tabu list. The use of tabu list prevents from returning to recently visited solutions, therefore it prevents from endless cycling and forces the search to accept even uphill moves. The length of tabu list (i.e., the tabu tenure) controls the memory of the search process. With small tabu tenures the search will concentrate on small area of the search

spaces. On the opposite, a large tabu tenure forces the search process to explore larger regions, because it forbids revisiting a higher number of solutions. (Blum and Roli, 2003)

3. *Aspiration criteria*. Tabu is sometimes too powerful, they may prohibit attractive moves, even when there is no danger of cycling, or they may lead to an overall stagnation of the searching process. It is thus necessary to use algorithmic devices that will allow one to revoke (cancel) tabus. These are called aspiration criteria. The simplest and most commonly used aspiration criteria consist in allowing a move even if it is tabu, if it results in a solution with an objective value better than of the current best-known solution (since the new solution has obviously not been previously visited). (Gendreau, 2002)
4. *Stopping criteria*. In theory, the search could go on forever, unless the optimal value of the problem at hand is known beforehand. In practice, obviously, the search has to be stopped at some point. The most commonly used stopping criteria in TS are:
    - after a fixed number of iterations (or a fixed amount of CPU time)
    - after some number of iterations without an improvement in the objective function value (the criterion used in most implementations)
    - when the objective reaches a pre-specified threshold value

    In complex tabu schemes, the search is usually stopped after completing a sequence of phases, the duration of each phase being determined by one of the above criteria. (Gendreau, 2002)
5. *Intermediate and long-term memory as a basis of strategies for Intensification and Diversification*. In fact, the fundamental elements of intensification and diversification strategies are already present in the short-term memory component of TS, since a short-term memory tabu list has an intensification role by temporarily locking in certain locally attractive attributes (those belonging to moves recently evaluated to be good), while it also has a diversification role by compelling new choices to introduce (or exclude) attributes that are not among those recently discarded (or incorporated). (Glover, 1990)

According to Crainic et al. (2005), two key concepts in TS are those of search intensification and search diversification. The idea behind search intensification is that regions of the search space that appear “promising” (usually because good solutions have been encountered close by) should be explored more thoroughly in order to make sure that the best solutions in these regions are found. Intensification is usually based on some type of intermediate-term memory, such as a recency memory, in which one would record the number of consecutive iterations that various “solution components” have been present without interruption in the current solution. Intensification is typically performed periodically restarting the search from the best currently known solution and by “freezing” (fixing) in this solution the components that seem more attractive. It is also often involves switching to a more powerful neighborhood operator for a short period of time.

Search diversification addresses the complementary need to perform a broad exploration of the search space to make sure that the search trajectory has not been confined to regions containing only mediocre solutions. It is thus a mechanism that tries to force the search trajectory into previously unexplored regions of the search space. Diversification is usually based on some form of long-term memory, such as frequency memory, in which one would record the total number of iterations (since the beginning of the search) that various “solution components” have been present in the current solution or have been involved in the selected moves (Crainic, et al., 2005).

The basic structure of TS (Glover, 1990) is depicted at Figure 3.9.

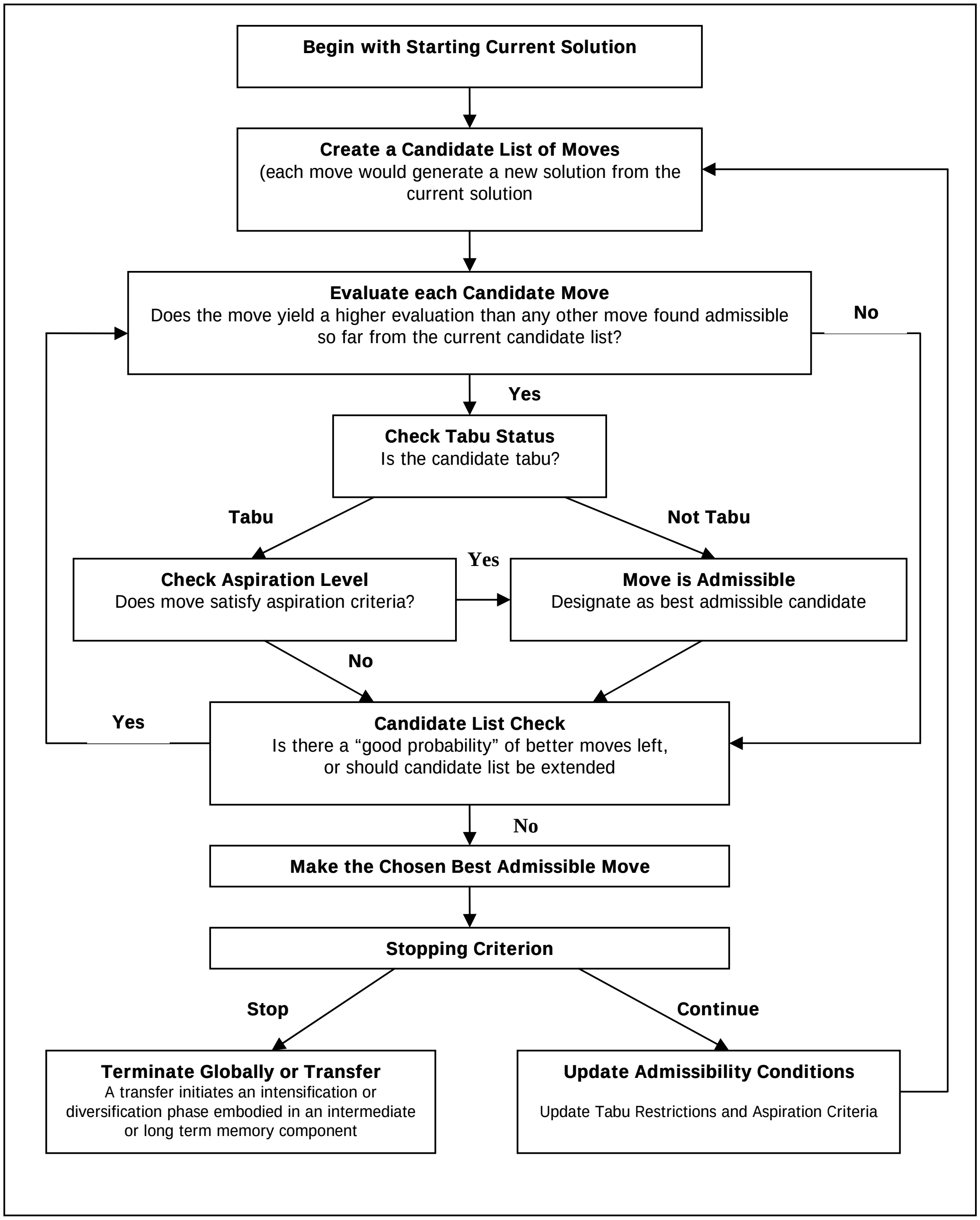

Begin with Starting Current Solution
Create a Candidate List of Moves
(each move would generate a new solution from the current solution
Evaluate each Candidate Move
Does the move yield a higher evaluation than any other move found admissible so far from the current candidate list?
No
Yes
Check Tabu Status
Is the candidate tabu?
Tabu
Not Tabu
Yes
Check Aspiration Level
Does move satisfy aspiration criteria?
Move is Admissible
Designate as best admissible candidate
No
Yes
Candidate List Check
Is there a "good probability" of better moves left, or should candidate list be extended
No
Make the Chosen Best Admissible Move
Stopping Criterion
Stop
Continue
Terminate Globally or Transfer
A transfer initiates an intensification or diversification phase embodied in an intermediate or long term memory component
Update Admissibility Conditions
Update Tabu Restrictions and Aspiration Criteria

Figure 3.9. Basic Tabu Search Algorithm

3.8 The Ant Colony Algorithm

3.8.1 Behavior of Real Ants

Ant colonies is a kind of social insect societies which presents a highly structured of social organization. Their self-organizing principles which allow the highly coordinated behavior of real ants can be exploited to coordinate populations of artificial agents that collaborate to solve computational problems.

Different aspects of the behavior of ants such as foraging (search or hunt for food and supplies), division of labor, brood sorting and cooperative transport inspired the different ant algorithms. In fact, ants coordinate their activities via *stigmergy*, where it is a form of indirect communication mediated by modifications of environment. Biologists have shown that stigmergic is often sufficient to be considered to explain how social insects can achieve self-organization.

In early research of ants' behavior, there is an important insight on the communication among individuals or between individuals and the environment, which is the use of chemicals called *pheromones* produced by ants. Trail pheromone is particularly important for some ant species such as Argentine ant *Iridomyrmex humilis* in the use for marking paths on the ground, for example, paths from food source to the nest. Thus, foragers can follow the path to food discovered by other ants just by sensing the pheromone trails. (Dorigo and Stutzle, 2004)

3.8.2 Double Bridge Experiments

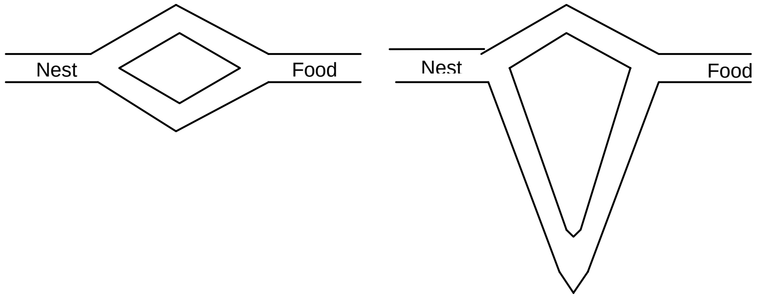
The foraging behavior of many ant species is based on the indirect communication mediated by pheromones. Ants deposit pheromones on the ground while walking from nest to food source and vice versa. The other ants will follow the path with pheromone trail and they tend to choose by probabilistically, the paths with higher amount of pheromones.

Several researches have run controlled experiments to investigate the pheromone trail-laying and pheromone trail-following behavior of ants. An outstanding experiment was designed and run by Deneubourg and colleagues (Deneubourg *et al.*, 1989), a double bridge was connected between a nest of ants of the Argentina species *Iridomyrmex humilis* and a food source. They varied the ratio $ratio = l_i / l_s$ between the length of two branches of the double bridge where l_i was the length of longer branch and l_s was the length of shorter branch.

In the first experiment, the two branches of the bridge had an equal length (*ratio* = 1: Figure 3.10(a)). The ants were left free to move between the nest and food source and the percentage of ants that chose one or the other of the two branches were observed over time. The outcome was eventually all the ants used the same branch.

The results can be explained as follows. There is no pheromone on the two branches when the trial is started and thus the ants chose any of the branches with equal probability. Yet, more ants will select one branch over the other due to random fluctuations. Since ants deposit pheromone while walking, the larger number of ants on a branch results in a larger amount of pheromone on that branch. This larger amount of pheromone will stimulate more ants to choose the branch again. Thus in the end, the ants converged to a single path (Figure 3.11(a)). This is an *autocatalytic* or positive feedback process by the local interactions among the individuals of the colony. (Dorigo and Stutzle, 2004)

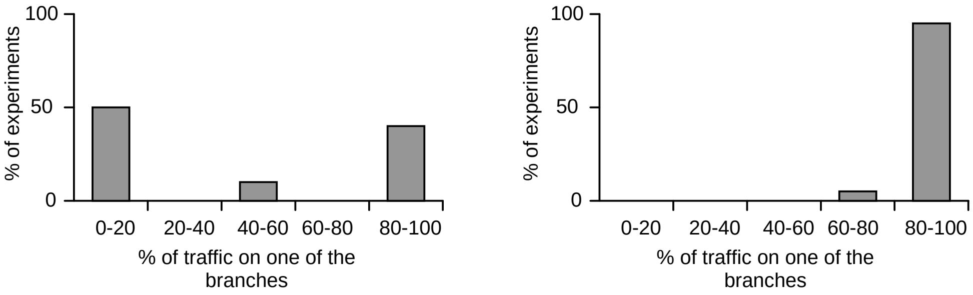
In the second experiment, the ratio is set to be *ratio* = 2 (Figure 3.10(b)) which means the long branch was twice as long as the short one. In this case, the ants converge to the short branch. As in the previous experiment, when the ants arrive at the decision point, they are free to choose any of the branches since there is no preference of pheromone trail on the two branches. It can be expected that half of the ants will go for short branch and another half will choose the long branch although random fluctuations may occur.



(a) (b)

Figure 3.10. Experimental setup for the double bridge experiment

However, there is difference in experiment setup between these two experiments, which is one branch was twice as long as the short branch in this second experiment. So the ants choosing the short branch will be the first to reach the food, when the ants reach the decision point in their way back to nest, they will see some pheromone trail on the short branch and therefore, they will choose the short branch with higher probability than the long branch. Hence pheromone starts accumulated faster on the short branch until eventually, all the ants end up using the short branch because of the autocatalytic process (Figure 3.11(b)).



(a) (b)

Figure 3.11. Results obtained with *Iridomyrmex humilis* ants in the double bridge experiment

Compared to the first experiment, the influence of initial random fluctuations is much reduced in the second experiment since the main mechanisms at work are the stigmergy, autocatalysis and differential path length. Besides, it is interesting that even though the long branch is twice as long as the short one, there were still got a small percentage of ants took the long branch. This may be interpreted as a type of "path exploration".

In an additional experiment, a new shorter connection between nest and food is offered after the ant colony is left to use the initial only long branch for 30 minutes (Figure 3.12). In this case, the ants were trapped in the long branch and the short branch was only selected sporadically.

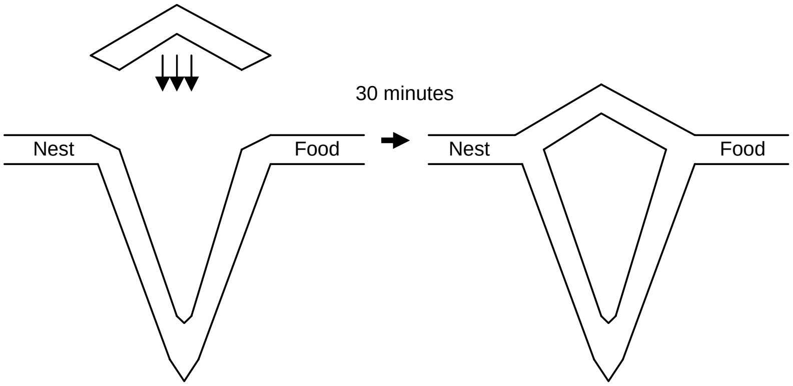


Figure 3.12. The initial experimental setup and the new situation after 30 minutes, when the short branch is added

This can be explained by the high concentration of pheromone on the long branch and the slow evaporation of pheromone. The ants will still choose the long branch even though short branch is offered due to the high concentration of pheromone on that long branch (Figure 3.13). This shows evaporation which allows the ant colony to "forget" the suboptimal path for exploration of new path is too slow.

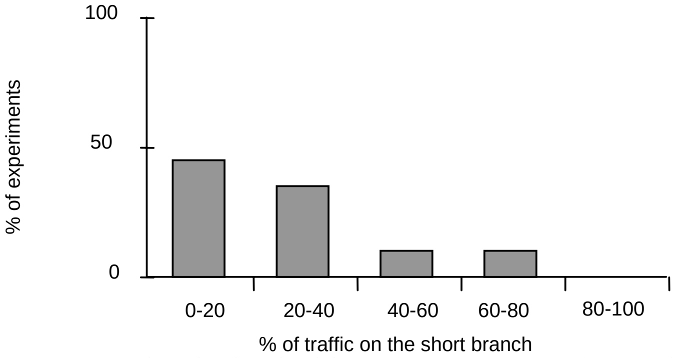


Figure 3.13. Results obtained with *Iridomyrmex humilis* ants in additional experiment

3.8.3 Artificial Ants

An artificial ant is an agent which moves from customer to customer on a graph. A variable $\tau(i, j)$ called the *artificial pheromone trail* is associated with each arc (i, j) on the graph, the phrase "pheromone" will be used as shorthand for this in the rest of the sections and chapters.

There are some major differences between real ants and artificial ants. Artificial ants have some memory to memorize the customers that had been served. They are not completely blind as real ants since artificial ants can determine how far the customers are. Besides, artificial ants live in an environment where time is discrete instead of continuous.

From real ants' behavior, there are three ideas that had been transferred to the artificial ants. Firstly, artificial ants prefer the paths with a high pheromone level. Secondly, amount of pheromone grow at the higher rate on shorter paths and thirdly, communication among the ants is mediated by the pheromone trail.

Basically, artificial ants work as follow; *m* artificial ants are positioned on some randomly selected nodes. As they move to a customer, they modify the pheromone trail on the edges used by local trail updating. After all the ants have completed their tour, artificial ant that gives the shortest tour modifies the edges that belonging to its tour by global trail updating. An example with artificial ants is shown as follow:

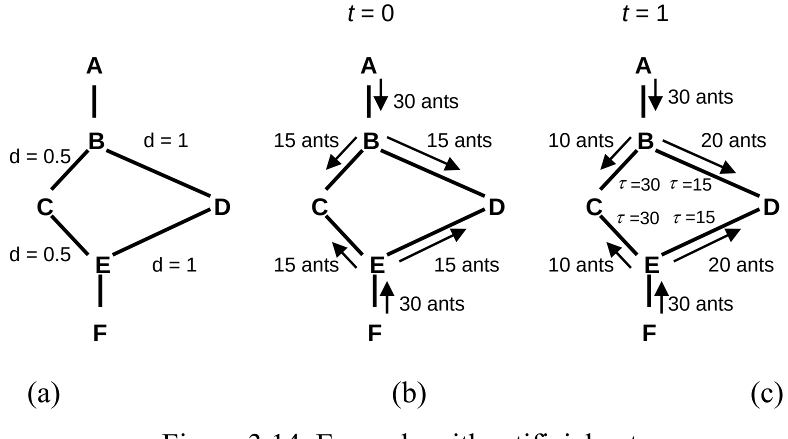


Figure 3.14. Example with artificial ants

Figure 3.14 (a) shows an initial graph with distances while Figure 3.14 (b) shows at time t = 0, there is no pheromone trail on edges, hence, 30 ants each at B and E choose to go opposite side whether through C or D with equal probability. So it is estimated that on average 15 ants will go toward C and 15 toward D respectively from B and E. Ants who follow the shorter path through C will reach at the opposite side faster than ants that follow the longer path through D. Artificial ants die once it complete its task (reach opposite side).

Thus, Figure 3.14 (c) shows at time t = 1 (assume that ant walks for 1 unit distance in 1 unit of time will deposit 1 unit of pheromone), 30 new ants that come from A to B will find a trail of intensity 15 on path that leads to D deposited by ants that went from B to D and a trail of intensity 30 on path that leads to C obtained as a sum of trail by those 15 ants went E from B via D and 15 ants went B from E via D. This process will continue until all the ants eventually follow the shortest path.

3.8.4 Ant System

Ant System which is a very simple ant-based algorithm is presented to illustrate the basic behavior of Ant Colony System. Initially, Ant System (AS) works as follows: m ants are initially positioned on n nodes chosen according to some initialization rule such as choosing by randomly, with at most one ant in each customer point. Each ant builds a tour incrementally by applying a state transition rule. Once the ants complete their tour, the pheromone on edges will be updated again by applying global updating rule (Dorigo and Caro, 1999).

An exploration mechanism is added to avoid a quick convergence of all the ants towards a sub-optimal path, which is artificial pheromone trails "evaporate" similar to real pheromone trails. The coefficient of pheromone decay parameter, ζ must be set as a value which less than 1. Thus, intensity of pheromone trails will decrease automatically and this favoring the exploration of other arcs during the whole search process (Dorigo and Caro, 1999).

Now, a similar process of real ants' behavior will be shown on how it can be put to work in a simulated world inhabited by artificial ants that try to solve the computational problem. There are *m* ants positioned on some randomly selected customers' points. In each time unit, each ants moves from one customer point to an un-served customer point. The time line is shown as below. (Dorigo *et al.*, 1996)

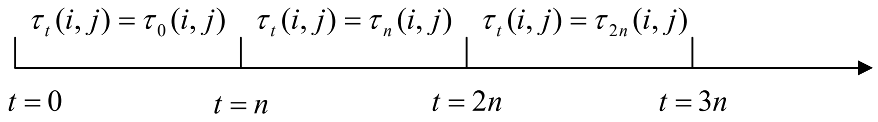

$\tau_t(i,j) = \tau_0(i,j)$
$\tau_t(i,j) = \tau_n(i,j)$
$\tau_t(i,j) = \tau_{2n}(i,j)$
$t = 0$
$t = n$
$t = 2n$
$t = 3n$

Figure 3.15**.** Time line in Ant System

Let $\tau_t(i,j)$ be the intensity of pheromone trail on edge (*i*, *j*) at time *t*. At time *t*, each of the ants will choose the next customer to serve. In other word, *m* moves had been done by *m* ants in time (*t*, *t* + 1). If this *m* moves is counted as one iteration in Ant System, then after *n* iterations (*t* = *n*), all ants have completed their tours respectively. At this point, the trail intensity is updated according to the formula below.

$$\tau_{t+n}(i,j) = (1-\zeta).\tau_t(i,j) + \zeta.\Delta\tau(i,j), \quad 0 < \zeta < 1 \tag{3.3}$$

$$\Delta\tau(i,j) = \sum_{k=1}^{m} \Delta\tau^k(i,j),$$

where ζ is pheromone decay parameter such that (1-ζ) is the evaporation of pheromone trail between time *t* and *t* + n. $\Delta\tau^k(i,j)$ is the quantity per unit of move of pheromone trail deposited by *k*-th ant between time *t* and *t* + n. According to the Ant-Cycle algorithm,

$$\Delta\tau^k(i,j) = \begin{cases} C/L_k & \text{if } (i,j) \text{ is on the complete tour of the } k\text{ - th ant in } (t, t+n) \\ 0 & \text{otherwise} \end{cases}.$$

where L_k is the tour length of the *k*-th ant.

In order to satisfy the constraint that all the customers had to be served, a data structure is associated for each ant called tabu list that saves the customers' points that had been served and forbid the ant from visiting it again before n iterations (a tour) is completed. After a tour is completed, the current solution of the ant is computed from its tabu list. Tabu list is then emptied and the ant can choose again the customer to serve. $Tabu_k(z)$ is defined as a dynamically growing vector that contains the tabu list, s-element of the list of k-th ant or in other word, s-th customer visited by k-th ant in the current tour.

Visibility $\eta(i,j)$ is a heuristic function, which was chosen to be the inverse of the distance between customer i and customer j, $1/c_{ij}$. The Euclidean distance c_{ij}, is the distance between customer i and customer j where $c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. This visibility will not be modified like pheromone during the run of Ant System.

The state transition probability for k-th ant from customer point i to customer point j is defined as follows

.

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_t(i,j)]^{\psi}.[\eta(i,j)]^{\beta}}{\sum\limits_{u \in J_k(i)} [\tau_t(i,u)]^{\psi}.[\eta(i,u)]^{\beta}} & \text{if } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where ψ and β are control parameters for the relative importance of the trail and visibility respectively, $\text{allowed}_k = N - tabu_k$, $N = \{1,2,...,n\}$ and $\text{tabu}_k = \{\text{tabu}_k(1), \text{tabu}_k(2), ..., \text{tabu}_k(e), ...\}$, $\text{tabu}_k(1)$ refers to the first customer served by the k-th ant and $\text{tabu}_k(e)$ refers to the e-th customer served by the k-th ant. $J_k(i)$ is the set of customers remain un-served by k-th ant after serves customer i.

3.8.5 Types of Ant Algorithm

Ant System was designed as a set of three ant algorithms where differing in the way the pheromone trail is updated by ants. Their names were ant-density, ant-quantity and ant-cycle (Dorigo and Caro, 1999).

3.8.5.1 Ant-Density Algorithm

Ant-Density algorithm uses local information, each ant deposits its trail at each step without waiting for the completion of tour construction. For Ant-Density algorithm, every time an ant goes from customer *i* to customer *j*, a quantity *C* of trail is left on edge (*i*, *j*).

$$\tau_{t+1}(i,j) = (1-\zeta).\tau_t(i,j) + \zeta.\Delta\tau(i,j), \quad 0 < \zeta < 1 \tag{3.5}$$

$$\Delta\tau(i,j) = \sum_{k=1}^{m} \Delta\tau^k(i,j),$$

$$\text{where } \Delta\tau^k(i,j) = \begin{cases} C & \text{if the } k\text{-th ant goes from } i \text{ to } j \text{ between } (t, t+1) \\ 0 & \text{otherwise} \end{cases}, \tag{3.6}$$

C is the constant related to the quantity of trail deposited by ants and ς is pheromone decay parameter.

3.8.5.2 Ant-Quantity Algorithm

Ant-Quantity algorithm works similar with Ant-Density algorithm where it uses local information and each ant deposits trail at each step without waiting for the end of the tour. Every time an ant goes from customer *i* to customer *j*, a quantity C / c_{ij} of trail is left on edge (*i*, *j*).

$$\tau_{t+1}(i,j) = (1-\zeta).\tau_t(i,j) + \zeta.\Delta\tau(i,j), \quad 0 < \zeta < 1$$

$$\Delta\tau(i,j) = \sum_{k=1}^{m} \Delta\tau^k(i,j),$$

where $$\Delta\tau^k(i,j) = \begin{cases} C/c_{ij} & \text{if the } k\text{-th ant goes from } i \text{ to } j \text{ between } (t, t+1) \\ 0 & \text{otherwise} \end{cases}, \qquad (3.7)$$

c_{ij} is the distance between customer *i* and customer *j*, *C* is the a constant related to the quantity of trail laid by ants and ς is pheromone decay parameter.

The different between these two algorithms is the increase of trail in Ant-Quantity algorithm is independent of c_{ij} while it is inversely proportional to c_{ij} in Ant-Density algorithm.

3.8.5.3 Ant-Cycle Algorithm

Ant-Cycle algorithm uses global information instead of local information. Artificial ants can forget part of the experience gained in the past move in order to obtain better exploration of new incoming global information.

The trail intensity is updated according to the formula after each of the ants has completed a tour.

$$\tau_{t+n}(i,j) = (1-\zeta).\tau_t(i,j) + \zeta.\Delta\tau(i,j), \quad 0 < \zeta < 1$$

$$\Delta\tau(i,j) = \sum_{k=1}^{m} \Delta\tau^k(i,j),$$

where

$$\Delta\tau^k(i,j) = \begin{cases} C/L_k & \text{if } (i,j) \text{ is on the complete tour of the } k\text{-th ant in } (t, t+n) \\ 0 & \text{otherwise} \end{cases} \qquad (3.8)$$

L_k is tour length of the *k* ant and ς is pheromone decay parameter.

Among these three ant algorithms, Ant-Cycle give the best-performing and it have later been inspiring a number of ant algorithms which including the Ant Colony Algorithm.

3.8.6 Our ACS Model for VRPSD

Ant System was efficient in discovering good or optimal solutions for small problems with nodes up to 30. But for larger problems, it requires unreasonable time to find such a good result. Thus, Dorigo and Gambardella (1997b) devised three main changes in Ant System to improve its performance which led to the existence of Ant Colony System.

Ant Colony System is different from Ant System in three main aspects. Firstly, state transition rule gives a direct way to balance between exploration of new edges and exploitation of a priori and accumulated information about the problem. Secondly, global updating rule is applied only to those edges which belong to the best ant tour and lastly, while ants construct the tour, a local pheromone updating rule is applied.

Basically, Ant Colony System (ACS) works as follows: *m* ants are initially positioned on *n* nodes chosen according to some initialization rule such as choosing by randomly, with at most one ant in each customer point. Each ant builds a tour incrementally by applying a state transition rule. While constructing the solution, ants also updating the pheromone on the visited edges by local updating rule. Once the ants complete their tour, the pheromone on edges will be updated again by applying global updating rule.

During construction of tours, ants are guided by both heuristic information and pheromone information. Heuristic information refers to the distances of the edges where ants prefer short edges. An edge with higher amount of pheromone is a desirable choice for ants. The pheromone updating rule is designed so that ants tend to leave more pheromone on the edges which should be visited by ants.

The Ant Colony System algorithm is given as follows (Dorigo and Gambardella, 1997b).

```
Initialize
Loop   //Each loop called an iteration
        Each ant is placed on a starting customer's point
        Loop   //Each loop called a step
                Each ant constructs a solution (tour) by applying a state
                transition rule and a local pheromone updating
        Until all ants have construct a complete solution.
        A global pheromone updating rule is applied.
Until stopping criteria is met
```

Figure 3.16. Ant Colony System's algorithm

3.8.7 Components of Ant Colony System

There are 3 main components which led to the definition of Ant Colony System; they are state transition rule, global updating rule and local updating rule. Each of these components will be shown in detail as follow.

3.8.7.1 Ant Colony System Transition Rule

In the ant ACS, an artificial ant k after serves customer r chooses the customer s to move to from set of $J_k(r)$ that remain to be served by ant k by applying the following state transition rule which is also known as *pseudo- random-proportional-rule*:

$$s = \begin{cases} \arg\max\limits_{u \in J_k(r)} \{[\tau(r,u)].[\eta(r,u)]^{\beta}\} & \text{if } a \le a_0 \quad \text{(exploitation)} \qquad (3.9) \\ S & \text{otherwise} \quad \text{(biased exploration)} \qquad (3.10) \end{cases}$$

where β is the control parameter of the relative importance of the visibility, $\tau(r,u)$ is the pheromone trail on edge (r, u) and $\eta(r,u)$ is a heuristic function which was chosen to be the inverse distance between customers r and u, a is a random number uniformly distributed in [0,1], a_0 $(0 \le a_0 \le 1)$ is a parameter and S is a random variable selected according to the probability distribution which favors edges which is shorter and higher amount of pheromone. It is same as in Ant system and also known as *random-proportional-rule* given as follow:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)].[\eta(r,s)]^{\beta}}{\sum\limits_{u \in J_k(r)} [\tau(r,u)].[\eta(r,u)]^{\beta}} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

where $p_k(r,s)$ is the probability of ant *k* after serves customer *r* chooses customer *s* to move to.

The parameter of a_0 determines the relative importance of exploitation versus exploration. When an ant after serves customer *r* has to choose the customer *s* to move to, a random number *a* ($0 \le a \le 1$) is generated, if $a \le a_0$, the best edge according to Equation (3.9) is chosen, otherwise an edge is chosen according to Equation (3.11) (Dorigo and Gambardella, 1997a).

3.8.7.2 Ant Colony System Local Updating Rule

While building a tour, ants visits edges and change their pheromone level by applying local updating rule as follow:

$$\tau(r,s) = (1-\rho).\tau(r,s) + \rho.\Delta\tau(r,s), \quad 0 < \rho < 1 \tag{3.11}$$

where ρ is a pheromone decay parameter and $\Delta\tau(r,s) = \tau_0$ (initial pheromone level).

Local updating makes the desirability of edges change dramatically since every time an ant uses an edge will makes its pheromone diminish and makes the edge becomes less desirable due to the loss of some of the pheromone. In other word, local updating drives the ants search not only in a neighborhood of the best previous tour.

3.8.7.3 Ant Colony System Global Updating Rule

Global updating is performed after all the ants have completed their tours. Among the tours, only the best ant which produced the best tour is allowed to deposit pheromone. This choice is intended to make the search more directed. The pheromone level is updated by global updating rule as follow:

$$\tau(r,s) = (1-\alpha).\tau(r,s) + \alpha.\Delta\tau(r,s), \quad 0 < \alpha < 1 \qquad (3.13)$$

where $\Delta\tau(r,s) = \begin{cases} 1/L_{gb} & \text{if } (r,s) \in \text{global} - \text{best} - \text{tour} \\ 0 & \text{otherwise} \end{cases}$

L_{gb} is the length of the global best tour from the beginning of the trial. (global-best) and α is pheromone decay parameter.

Global updating is intended to provide greater amount of pheromone to shorter tours. Equation (3.13) dictates that only those edges belong to globally best tour will receive reinforcement.

3.8.8 Ant Colony System Parameter Settings

From the experiments done by previous researchers, the numerical parameters are set as following values:

$\beta = 2,\ a_0 = 0.9,\ \alpha = \rho = 0.1,\ \tau_0 = (n.L_{nn})^{-1}$,

where L_{nn} is the tour length produced by the nearest neighbor heuristic and n is the number of customers. The number of ants used is m = 10. These values were obtained by a preliminary optimization phase in which it is found that the experimental optimal values of the parameters were largely independent of the problem except for τ_0 (Dorigo and Gambardella, 1997b).

3.8.9 Pheromone Behavior and Its Relationship to Performance

Pheromone is the chemical substance from ants to leave the trail for other ants to follow from the nest to the food source. In reality, the more of the times that an edge is visited by ants, the greater amount of pheromone will be leave on the visited edges to direct the subsequent ants, eventually, all the ants end up with using the shorter path.

But in ACS, the application of local updating rule in Equation (3.12) makes their pheromone diminish, which makes them less attractive. Therefore, ants favoring the exploration of edges that not yet visited. As a consequence, ants never converge to a common path. The reason behind is given that if ants explore different paths, then there is higher probability that one of them will find an improving solution instead of all the ants converge to the same tour.

3.8.10 Ant Colony System with Local Search

Heuristic approaches to the tour obtained by ants can be classified as tour constructive heuristic. Tour constructive heuristic usually start selecting randomly a customer point and build the feasible solution piece-by piece by adding new customers' points chosen according to the selected heuristic rule.

There is a kind of local optimization heuristic called tour improvement heuristics (Dorigo and Gambardella, 1997b). From a given tour, tour improvement heuristics attempt to reduce total distance of the tour by exchanging the edges chosen according to some heuristic rule until a local minimum is obtained. In general, tour improvement heuristics produce a better quality results than constructive heuristics. A general approach is to use constructive heuristics to generate a solution and then optimize locally the solution using tour improvement heuristics. To improve the performance of ACS, tour improvement heuristics had been inserted into ACS.

The local search that used in this study is the Descent Method. Descent method is known as hill-climbing heuristic or greedy heuristic. It is an algorithm that always takes the best immediate or local solution while finding an answer. It proceeds by examining some neighborhoods of the current solution. This process is repeated until there is no further improvement and the final solution is taken as the solution of the problem.

The general procedure of Descent method for minimizing problem is as follow:

Step1: Select an initial solution; say $x \in S$ (where *S* is the set of feasible solutions).

Step 2: Choose a solution $x^{'} \in N(x)$ such that $F(x^{'}) < F(x)$ (where $N(x)$ is the neighborhood of *x*).

If there is no such $x^{'}$, x is considered as a local optimum and the method stops.

Else set $x = x^{'}$ and repeat step 2.

The neighborhood of x is defined as its associated set of neighboring points which obtained directly from x by a symmetric operation called move. The move or operation used in obtaining the neighborhood of a solution x is swapping two positions of customers' points in x.

Thus, the ACS with local search Descent Method algorithm is as follow.

```
Initialize
Loop   //Each loop called an iteration
       Each ant is placed on a starting customer's point
       Loop   //Each loop called a step
              Each ant constructs a solution (tour) by applying a
              state transition rule and a local pheromone updating
       Until all ants have construct a complete solution.
       Each ants is brought to a local minimum by a tour
       improvement heuristic
       A global pheromone updating rule is applied.
Until stopping criteria is met
```

Figure 3.17. ACS with local search Descent Method algorithm

Thus, the complete algorithm together with the flow for the method of study is summarized as follow.

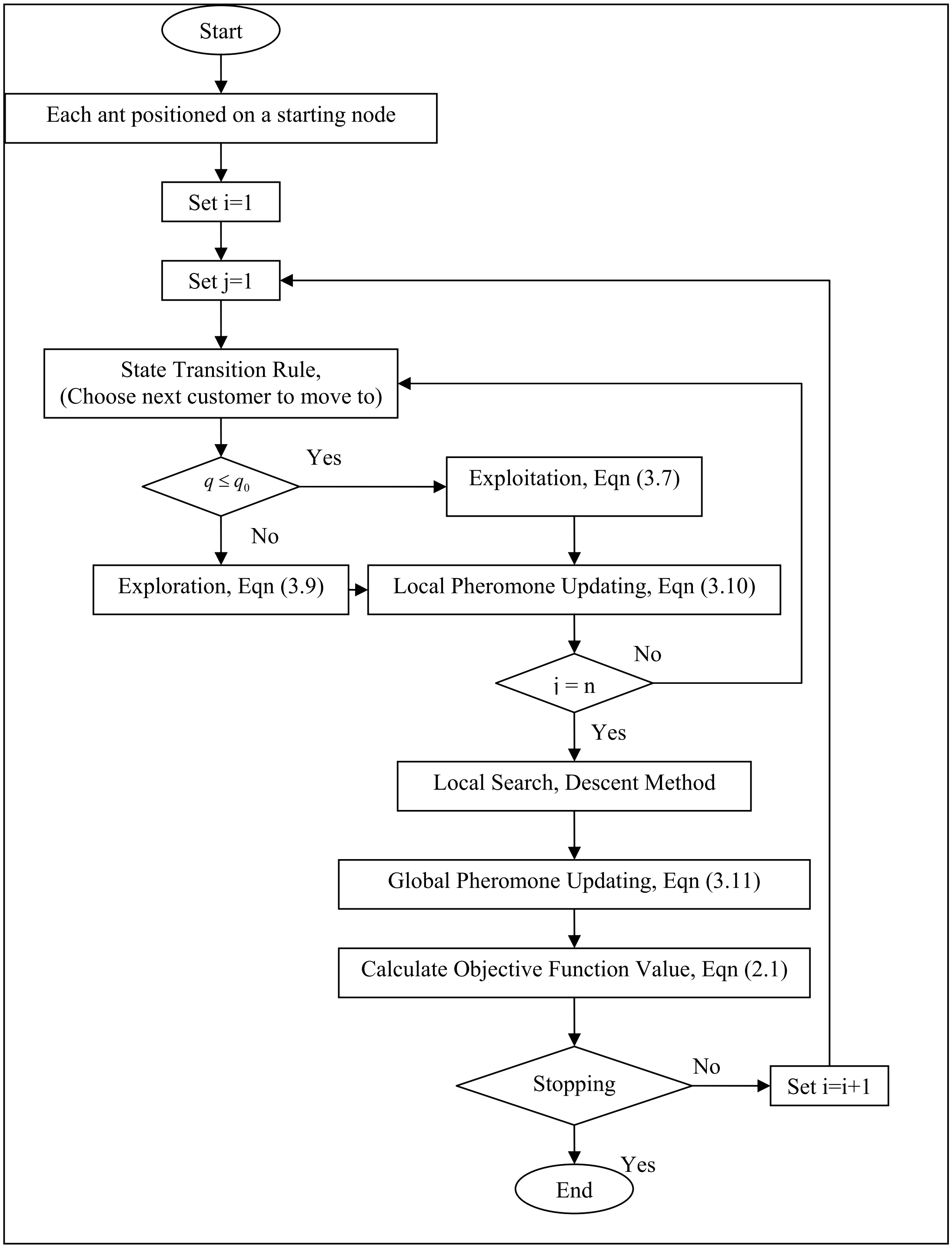

Start
Each ant positioned on a starting node
Set i=1
Set j=1
State Transition Rule,
(Choose next customer to move to)
$q \leq q_0$
Yes
No
Exploitation, Eqn (3.7)
Exploration, Eqn (3.9)
Local Pheromone Updating, Eqn (3.10)
i = n
No
Yes
Local Search, Descent Method
Global Pheromone Updating, Eqn (3.11)
Calculate Objective Function Value, Eqn (2.1)
Stopping
No
Set i=i+1
Yes
End

Figure 3.18. Flow chart of ACS implemented

3.9 Simulated Annealing

Simulated annealing is a stochastic approach for solving combinatorial optimization problems (Hillier and Lieberman, 2005). It uses local search (iterative improvement) technique that allowed accepting non-improving neighboring solutions to avoid being trapped at a poor local optimum with a certain probability.

It starts with a feasible initial solution (which is defined as current solution, x) and uses a move selection rule to select the next solution (which is called as neighboring solution, $x^{'}$) from the neighborhood of current solution. In this study, the neighboring solution is obtained by switching two customers' points chosen randomly. If the neighboring solution is better than the current solution, it is accepted to be the next solution. If it is worse than current solution, then it is accepted as the current solution with some probability depends on how worse it is to allow the search to escape a local optimal solution.

The probability of acceptance is calculated based on Boltzmann distribution shown as follow (Kirkpatrick *et al*., 1983).

$$P(\delta) = \exp(-|\delta| / T_k) \tag{3.14}$$

where T_k is the current temperature and δ is the change in total cost (i.e. the cost of the neighboring solution minus the cost of the current solution, $F(x^{'}) - F(x)$). If $a < P(\delta)$ where a is the randomly generated number between 0 and 1, the non-improving neighboring solution $x^{'}$ is accepted as the current solution. In other words, the move selection rule usually will accept a step that is only slightly downhill, but seldom will accept a steep downward step.

The probability of accepting a non-improving solution is controlled by two factors which are the different of the objective functions and the temperature. The higher the value of δ, the lower is the probability to accept the non-improving solution as current solution. On the other hand, with a higher T_k, the higher is the probability for accepting the non-improving solution as the current solution.

The value of temperature varies from a relatively large value to a small value close to zero. These values are controlled by a cooling schedule that specifies the initial and incremental temperature values at each stage of the algorithm. One common example for cooling schedule is as follow.

$$T_{k+1} = \gamma \times T_k$$

where T_k is the current temperature and γ is the cooling rate. The temperature will be reduced after a certain number of moves.

Lastly, iterations of SA stop when the stopping criterion is being met. There are a few types of stopping criteria. For example, when the desired number of iterations has been performed or when the iteration has been performed at the smallest value of *T* in the temperature schedule, the best trial solution is accepted as the final solution.

The illustration of the SA algorithm which will be used in this study is as follows.

```
Step 1: Generate an initial solution, $x$ by using perturbation.
        Determine the total distance for the tour, $F(x)$.
Step 2: Set the initial temperature parameter $T_{TCC}$ where $TCC = 0$.
        Set $i = 1, j = 1$.
        Define cooling rate, $\gamma$ and number of maximum iteration.
        Set repetition counter = $RC$.
        Set Best Solution = $x$ and Best Distance = $F(x)$.
Step 3: Choose two cities randomly and swap the two cities in $x$
        to form a new tour, $x'$.
        Determine the total distance for the tour, $F(x')$.
Step 4: If $F(x')$ < Best Distance,
            Update Best Solution = $x'$ and
            Best Distance = $F(x')$
        If $F(x') < F(x)$,
            Update $x = x'$, $F(x) = F(x')$
```

Else check the probability of acceptance,

If random number (0,1), $a < \exp\left[\frac{-|\delta|}{T_{TCC}}\right]$

Update $x = x^{'}$, $F(x) = F(x^{'})$

Step 5: If *i* = number of maximum iterations

Stop the iteration.

Set the best solution as final solution.

Else set *i* = *i* + 1 and go to step 6.

Step 6: If $j = CR$

Reset *j* = 1,

Update *TCC* = *TCC* + 1 and temperature by cooling schedule

$T_{TCC} = \gamma \times T_{TCC-1}$ and go to step 3.

Else *j* = *j* + 1 and go to step 3.

Illustration of Ant Colony Algorithm

The procedure and calculations for ACS algorithm will be shown in detail through a numerical example with a small size of problem. Each of the steps will be clearly shown in both sections.

Example 3.1: Suppose that there are 5 customers and the data set is shown in Table 3.2.

Table 3.2. Data set for the numerical example

Nodes	**Coordinate *x***	**Coordinate *y***	**Demands**
Depot	14	11	-
1	8	12	0 – 6
2	11	18	0 – 5
3	20	13	2 – 5
4	14	5	2 – 4
5	20	2	2 – 6

The parameter setting for this example is shown in Table 3.3.

Table 3.3. Parameter settings for numerical example

Number of ants	2
Capacity of vehicle	10
a_0	0.7
α, ρ	0.1
β	2
Penalty for route failure, *b*	2
Maximum iterations (ACS)	10
Num. of consecutive non-improving solutions allowed (ACS)	5

STEP 1: Initial Pheromone Declaration

L_{nn} is the tour length produced by the nearest neighbor heuristic. For this example,

$L_{nn} = L_{55}$= 46.7948 units,

$\tau_0 = (n.L_{nn})^{-1} = (5 \times 46.7948)^{-1} = 0.004274$ units.

STEP 2: Generate starting position for each ants.

Ants' starting positions are generated randomly by computer and it can be represented by notations below:

$m = 2,\ m_{1,0} = 1,\ m_{2,0} = 4.$

STEP 3: Choose next customer point to be visited

Both ants have to build their tour incrementally as follow.

Iteration 1: (first customer point to be visited)

For first ant, the random number generated by computer, $a = 0.1109 < 0.7$, it will search for next customer location *s* to move to by Exploitation using Equation (3.9).

$s = \arg\max_{u \in J_k(r)} \{[\tau(r,u)].[\eta(r,u)]^{\beta}\}$, where $J_1(1) = \{0, 2, 3, 4, 5\}$ and

$$\begin{aligned}\max\{[\tau(r,u)].[\eta(r,u)]^{\beta}\} &= \max\{[\tau(1,0)].[\eta(1,0)]^2, [\tau(1,2)].[\eta(1,2)]^2, [\tau(1,3)].[\eta(1,3)]^2,\\ &\qquad [\tau(1,4)].[\eta(1,4)]^2, [\tau(1,5)].[\eta(1,5)]^2\}\\ &= \max\{0.0001155, 0.000095, 0.000029, 0.00005, 0.000018\}\end{aligned}$$

$$= 0.0001155$$

Thus, $s = 0$ or $m_{1,1} = 0$

For the second ant, $a = 0.8337 > 0.7$, it will search for next customer location s to move to by Exploration using Equation (3.11).

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)].[\eta(r,s)]^\beta}{\sum\limits_{u \in J_k(r)} [\tau(r,u)].[\eta(r,u)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases}$$

where $J_2(4)$ = {0, 1, 2, 3, 5} and

$$\sum_{u \in J_k(r)} [\tau(r,u)].[\eta(r,u)]^\beta = [\tau(4,0)].[\eta(4,0)]^2 + [\tau(4,1)].[\eta(4,1)]^2 + [\tau(4,2)].[\eta(4,2)]^2$$

$$+ [\tau(4,3)].[\eta(4,3)]^2 + [\tau(4,5)].[\eta(4,5)]^2$$

$$= 0.0001187 + 0.00005 + 0.000024 + 0.0000427 + 0.000095$$

$$= 0.0003304$$

Thus, $P_2(4,0) = 0.0001187 / 0.0003304 = 0.35926$

$P_2(4,1) = 0.15133$

$P_2(4,2) = 0.07264$

$P_2(4,3) = 0.12924$

$P_2(4,5) = 0.28753$

A line is drawn based on the accumulation of probability as in Figure 3.18.

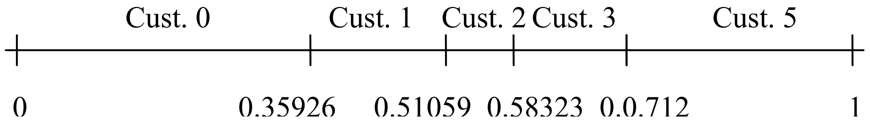


Figure 3.19. Roulette wheel for selecting next customer to move to

A random number generated, a = 0.695425 falls in the forth interval which represents customer 3. Thus, s = 3 or $m_{2,1}$ = 3.

STEP 4: Local pheromone updating

After all ants move to their next customers' points, the pheromone on edges that the ants pass by will be updated by local pheromone updating rule using Equation (3.12) as follow.

$$\tau(r,s) = (1-\rho).\tau(r,s) + \rho.\Delta\tau(r,s), \quad 0 < \rho < 1$$

For first ant,

$$\tau(1,0) = (0.9)\times\tau(1,0) + 0.1\times\tau_0, \qquad \rho = 0.1$$

$$= (0.9)\times 0.004274 + (0.1)\times 0.004274$$

= 0.004274 units.

Pheromone on edge (4, 3) in which second ant had passed by is updated by the same way as above. The initial pheromone for all edges is τ_0, thus, first round of local pheromone gives no changes for the pheromones on edges. The pheromones will be various for all edges after global pheromone updating rule is applied.

STEP 3 is followed by STEP 4 are repeated for *n* times until all ants have built their complete tour respectively. The complete tours for both ants after *n* iterations for STEP 3 followed by STEP 4 are as follow.

Ant 1: 1 - 0 - 4 - 5 - 2 - 3 - 1

Ant 2: 4 - 3 - 0 - 1 - 2 - 5 - 4

Before the total expected costs for all tours are evaluated, the customers' points sequence are rearranged such that they start and end at node 0 as follow for total expected cost calculation purpose.

Ant 1:	0 - 4 - 5 - 2 - 3 - 1 - 0,	Total expected cost = 59.8349 units
Ant 2:	0 - 1 - 2 - 5 - 4 - 3 - 0,	Total expected cost = 56.5243 units

STEP 5: Local Search (Descent Method)

After all ants have completed their tour, all ants will be brought to a local minimum by a local search called Descent Method. For each of the iteration, two customers' points are randomly chosen for swapping their positions in the given sequence in accepted tour and its total expected cost is computed. The iterations will be stopped after 10 consecutive non-improving solutions and the final best solution is recorded. Local minima for both ants are as follow.

Ant 1: 0 - 4 - 5 - 3 - 2 - 1 - 0,	Total expected cost = 54.6327 units
Ant 2: 0 - 2 - 1 - 4 - 5 - 3 - 0,	Total expected cost = 55.0749 units

STEP 6: Global pheromone updating

The global best tour is the globally best tour from the beginning of the trial, the one and only ant that produces the global best tour is allowed to deposit the pheromone on the edges that belonging to that global best tour using Equation (3.13) as follow.

$$\tau(r,s) = (1-\alpha).\tau(r,s) + \alpha.\Delta\tau(r,s), \quad 0 < \alpha < 1$$

where $\Delta\tau(r,s) = \begin{cases} 1/L_{gb} & \text{if } (r,s) \in \text{global} - \text{best} - \text{tour} \\ 0 & \text{otherwise} \end{cases}$

Global best tour = 0 - 4 - 5 - 3 - 2 - 1 - 0, Total expected cost = 54.6327 units

$$\begin{aligned} \tau(0,4) &= (0.9)\times\tau(0,4) + (0.1)\times(1/54.6327), \quad \alpha = 0.1 \\ &= (0.9)\times 0.004274 + (0.1)\times(1/54.6327) \\ &= 0.005677 \end{aligned}$$

By the same way, the rest of $\tau(4,5), \tau(5,3), \tau(3,2), \tau(2,1), \tau(1,0)$ can be obtained.

After global pheromone updating is applied, one loop or iteration is said to be completed. The next iteration is started again from STEP 2 to STEP 6 until the stopping criteria is met. Stopping criteria for this example is either obtained 5 consecutive non-improving solutions or the iterations reach the predetermined number of maximum iteration depending on which criteria is being met first.

For this example, the iterations stop after 7 iterations. The final solution is 0 - 1 - 2 - 3 - 5 - 4 - 0 with Total Expected Cost = 54.4587 units. The complete result can be found in Appendix A.

llustration of the Calculation of VRPSD Objective Function

Example 3.2 : Consider 5 customers where customers' coordinates and demands, and its distance matrix are shown at Table 3.2 and 3.3, respectively. Set route to be follows is 0 → 2 → 4 → 5 → 3 → 1 → 0 as described in Figure 3.19.

Table 3.4. Eample of customer coordinates and demands

I	**x**	**y**	**demand**
0	50	50	0
1	82	23	U(1,3)
2	21	63	U(2,6)
3	79	46	U(0,2)
4	6	80	U(0,2)
5	35	33	U(1,3)

Table 3.5. Distance matrix:

Node	0	1	2	3	4	5
0	0	42	32	29	53	23
1	42	0	73	23	95	48
2	32	73	0	60	23	33
3	29	23	60	0	81	46
4	53	95	23	81	0	55
5	23	48	33	46	55	0

Let *f* = 1.1, then the quantity of garbage is

$$Q = \frac{\sum_{n} E(\xi_i)}{f} = \frac{(5)([2+4+1]/3)}{1.1} = 11\text{, vehicle's quantity } Q = 11$$

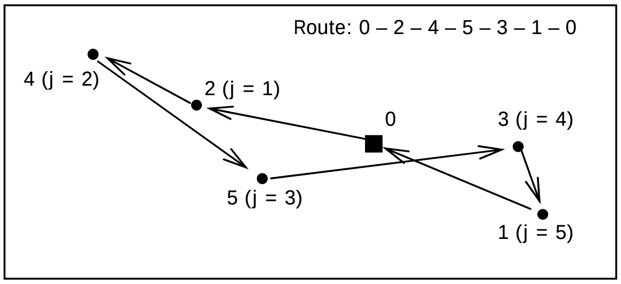


Figure 3.20. 5 customers network (depot excluded)

On above figure, j represents the order/ sequence to be visited. For example: j = 1 means 1st location to be visited, that is node 2. j = 2 means 2nd visited, that is node 4, and so on.

Iteration 1 : *j* = 5

Set $f_{j=5}(q) = c_{j=5,0}$ = 42

note that j = 5 doesn't mean that node 5, but the customer no 5th visited, that is node 1. So f_5 represents travel cost from node 1 and node 0 that is 42.

Iteration 2 : *j* = 4

j = 5-1 = 4 → 4th visited → node 3,

a. Calculate $f_4^r(q)$ based on Eq. 2.3.

$$f_j^r(q) = c_{j,0} + c_{0,j+1} + \sum_{k=1}^{K} f_{j+1}(Q - \xi^k) p_{j+1,k}$$

$$f_4^r(q) = c_{4,0} + c_{0,4+1} + \sum_{k=1}^{3} f_{4+1}(11 - \xi^k) p_{4+1,k} \ ,$$

where

- $c_{4,0}$ → travel cost from node 4th visited (node3) to node 0 is 29 (see distance matrix)
- $p_{5,k}$ represents probability for node 5th visited (node 1). Node 1 follows U(1,3), so k = 1, 2, 3. $p_{5,k}$=1/3, since there are 3 outcomes (1,2 and 3)

for the 2nd iteration, set f_5 (q) = 42 for all q.

thus: $$f_4^r(q) = 29 + 42 + p_{5,k}[f_5(11-1) + f_5(11-2) + f_5(11-3)]$$

$$= 29 + 42 + \tfrac{1}{3}[f_5(10) + f_5(9) + f_5(8)]$$

$$= 29 + 42 + \tfrac{1}{3}[(3)x(f_5)]$$

$$= 29 + 42 + \tfrac{1}{3}[(3)x(42)] = 113$$

The value $f_4^r(q)$ is the same for all q, $f_4^r(q)$ = 113.

b. Calculate $f_4^p(q)$ based on Eq. 2.2

set $b = 0$, $f_4^p(11) = c_{4,5} + \sum_{k=1}^{3} f_{4+1}(11-k)p_{4+1,k}$

$$= 23 + \{[f_5(11-1)p_{5,1}] + [f_5(11-2)p_{5,21}] + [f_5(11-3)p_{5,1}]\}$$

$$= 23 + \tfrac{1}{3}\{[f_5(10)] + [f_5(9)] + [f_5(8)]\}$$

$$= 23 + (\tfrac{1}{3})(3)(42) \ = 65$$

$f_4^p(11) = f_4^p(10) = ... = f_4^p(3) = 65$

$$f_4^p(2) = c_{4,5} + \sum_{k=1}^{2} f_5(2-k)p_{5,k} + \sum_{k=3}^{3} 2c_{5,0} + f_5(2+11-k)p_{5,k}$$

$$= 23 + \{[f_5(11-1)p_{5,1}] + [f_5(11-2)p_{5,2}]\} + [(2).(42) + f_5(2+11-3)p_{5,3}]$$

$$= 23 + \tfrac{1}{3}\{[f_5(10)] + [f_5(9)] + 84 + [f_5(10)]\}$$

$$= 23 + \tfrac{1}{3}\{42 + 42 + 84 + 42\}$$

$$= 93$$

$$f_4^p(1) = c_{4,5} + \sum_{k=1}^{1} f_5(1-k)p_{5,k} + \sum_{k=2}^{3} 2c_{5,0} + f_5(1+11-k)p_{5,k}$$

$$= 23 + \{[f_5(1-1)p_{5,1}]\} + \{[2c_{5,0} + f_5(12-2)p_{5,2}] + [2c_{5,0} + f_5(12-3)p_{5,3}])\}$$

$$= 23 + \tfrac{1}{3}\{[f_5(0)] + [(2)(42) + f_5(11) + (2)(42) + f_5(10)]\}$$

$$= 23 + \tfrac{1}{3}\{42 + [(2)(42) + 42 + (2)(42) + 42]\}$$

$$= 120.9$$

$$f_4^p(0) = c_{4,5} + \sum_{k=1}^{3} 2c_{5,0} + f_5(0+11-k)p_{5,k}$$

$$= 23 + \{[2c_{5,0} + f_5(11-1)p_{5,1}] + [2c_{5,0} + f_5(11-2)p_{5,2} + [2c_{5,0} + f_5(11-3)p_{5,3}])\}$$

$$= 23 + \tfrac{1}{3}\{[f_5(10)] + [(2)(42) + f_5(11) + (2)(42) + f_5(10)]\}$$

$$= 23 + \tfrac{1}{3}\{42 + [(2)(42) + 42 + (2)(42) + 42]\}$$

$$= 149$$

c. Compare $f_4^r(.)$ with $f_4^p(q)$ for finding threshold h_4 and $f_4(q)$.

h_4 is the last value of q when $f_4^r(.) \ \leq \ f_4^p(q)$, so h_4=2.

$f_4(q) = \min\{f_4^p(q), f_4^r(q)\}$

thus $f_4(11) = f_4(10) = ... = f_4(3) = 65$, $f_4(2)$=93, $f_4(1)$=113, $f_4(0)$=113

These values of $f_4(q)$ will be used for the next iteration.

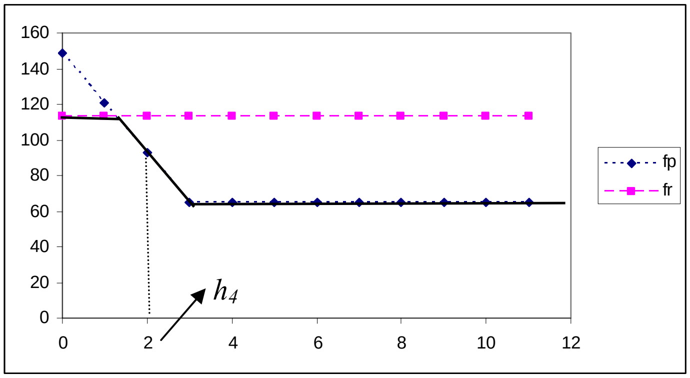


Figure 3.21. Illustration of h_j determination

Iteration 3 : *j* = 3

j = 5-2 = 3 → 3rd visited → node 5,

a. Calculate $f_3^r(q)$

$$f_3^r(q) = c_{3,0} + c_{0,3+1} + \sum_{k=0}^{2} f_{3+1}(11-k)p_{3+1,k} \ ,$$

where

- $c_{3,0}$ → travel cost from node 3th visited (node5) and node 0 is 23 (see distance matrix)
- $p_{4,k}$ represents probability for node 4th visited (node 3). Node 3 follows U(0,2), so k = 0,1, 2. $p_{4,k}$ =1/3, since there are 3 outcomes (0,1,2)

thus: $f_3^r(q) = 23 + 29 + p_{4,k}[f_4(11-0) + f_4(11-1) + f_4(11-2)]$

$= 23 + 29 + \frac{1}{3}[f_4(11) + f_4(10) + f_4(9)]$, use result from 2nd iteration for $f_j(q)$ values

$= 52 + \frac{1}{3}[65 + 65 + 65]$

$= 117$

The value $f_3^r(q)$ is the same for all q, $f_3^r(q)$ = 117.

b. Calculate $f_3^p(q)$

$$f_3^p(11) = c_{3,4} + \sum_{k=0}^{2} f_{3+1}(11-k)p_{3+1,k}$$

$$= 46 + \{[f_4(11-0)p_{4,1}] + [f_4(11-1)p_{4,2}] + [f_4(11-2)p_{4,3}]\}$$

$$= 46 + \tfrac{1}{3}\{[f_4(11)] + [f_4(10)] + [f_4(9)]\}$$

$$= 46 + (\tfrac{1}{3})(65+65+65) = 111$$

$$f_3^p(11) = f_3^p(10) = ... = f_3^p(2) = 111$$

$$f_3^p(1) = c_{3,4} + \sum_{k=0}^{1} f_4(1-k)p_{4,k} + \sum_{k=2}^{2} 2c_{4,0} + f_4(1+11-k)p_{1,k}$$

$$= 46 + \tfrac{1}{3}\{[f_4(1) + f_4(0)] + [(2)(29) + f_4(10)]\}$$

$$= 46 + \tfrac{1}{3}\{113 + 113 + 58 + 65\}$$

$$= 162.33$$

$$f_3^p(0) = c_{3,4} + \sum_{k=0}^{0} f_4(0-k)p_{4,k} + \sum_{k=1}^{2} 2c_{4,0} + f_4(0+11-k)p_{4,k}$$

$$= 46 + \tfrac{1}{3}\{[f_4(0)] + [(2)(2)(29) + f_4(10) + f_4(11)]\}$$

$$= 46 + \tfrac{1}{3}\{113 + 116 + 65 + 65\}$$

$$= 165.67$$

c. Compare $f_3^r(.)$ with $f_3^p(q)$ for finding threshold h_3 and $f_3(q)$.

h_3 is the last value of q when $f_3^r(.) \le f_3^p(q)$, h_3=2.

$$f_3(q) = \min\{f_3^p(q), f_3^r(q)\}$$

so $f_3(11) = f_3(10) = ... = f_3(2) = 111$, $f_3(1)$=162.33, $f_3(0)$=165.67

These values of $f_3(q)$ will be used for the next iteration.

These procedures are repeated until $f_0(Q)$ is found. The $f_0(Q)$ = 221 and the values of threshold are h_4 = 2, h_3 = 2, h_2 = 2 and h_1 = 1.

CHAPTER 4

TABU SEARCH FOR SINGLE VRPSD

4.1 Introduction

In this section we give brief description of the Tabu Search algorithm that was implemented on the single VRPSD, followed by the experimental results and discussion.

4.2 Tabu Search for Single VRPSD

4.2.1 Initial Solution

It is necessary to generate a number of solutions to initialize the main search process. The choice of the initial solution is well known to be important, usually if the initial solution is quite good, it will leads to faster convergence to the optimum solution. Here two simple and fast methods were used to generate initial solution: Nearest Neighbour (NN) and Farthest Insertion (FI) heuristics. The best among them was chosen as the starting solution to the main body of the search itself, based on which heuristic that yields lower cost from the calculation of VRPSD objective function. The first node to be visited from the depot is the node that has largest demand.

a. Nearest Neighbour

The nearest neighbour algorithm is easy to implement and executes quickly, but usually not the optimal one. It can sometimes miss shorter routes which are easily noticed with human insight, due to its "greedy" nature.

These are the steps of the algorithm:

1. Start from the depot, form tour with next depot to be visited is node that has largest demand.
2. Scan remaining cities for the city nearest to last node added in the tour. Add this node to the existing tour.
3. Repeat step 2 until all nodes are added. Close path to form a tour.

b. Farthest Insertion

These are the steps of the algorithm:

1. Start with a subtour made of the depot and the largest demand node.
2. Select a node *k* that farthest from largest demand node. Add this node to form subtour depot - largest demand node - *k* – depot.
3. Scan remaining nodes for the node *k* farthest from any nodes in tour.
4. Find edge (*i*,j) that minimize $c_{ik} + c_{jk} - c_{ij}$ where *i* and *j* are in the tour. Insert *k* between *i* and *j*, except for edge (*i*,*j*) that represents (depot, largest demand node).
5. Repeat step 3 and 4 until all cities are in the tour.

4.2.2 Neighbourhood Exploration

The neighbourhood of a solution contains all solutions that can be reached by applying two local search methods, namely 2-opt and modified Lin-Kernighan (LK). LK algorithm was published in 1973, and from 1973 to 1989 this was the “world champion” heuristic. An LK search is based on 2-Opt moves, although a significantly restricted subset of them and it has many ideas in common with tabu search.

The operation of the inner loop of the LK algorithm is as follows:

1. Start with a tour 1, 2,… *N*.
2. If the cost of the tour 1, 2,… *N,*1 is better than the best tour found so far, make it the best tour.
3. Add an edge (*N,k*) where 1≤*k*<(*N*-1) to make tour into *δ*-tour.
4. Add the edge (*N,k*) to the added-list.
5. Remove the edge (*k,k+1*). Note that the last part of the new tour is traversed backwards from the old tour.
6. Add the edge (*k*,*k*+1) to the deleted-list.
7. Go back to step 3.

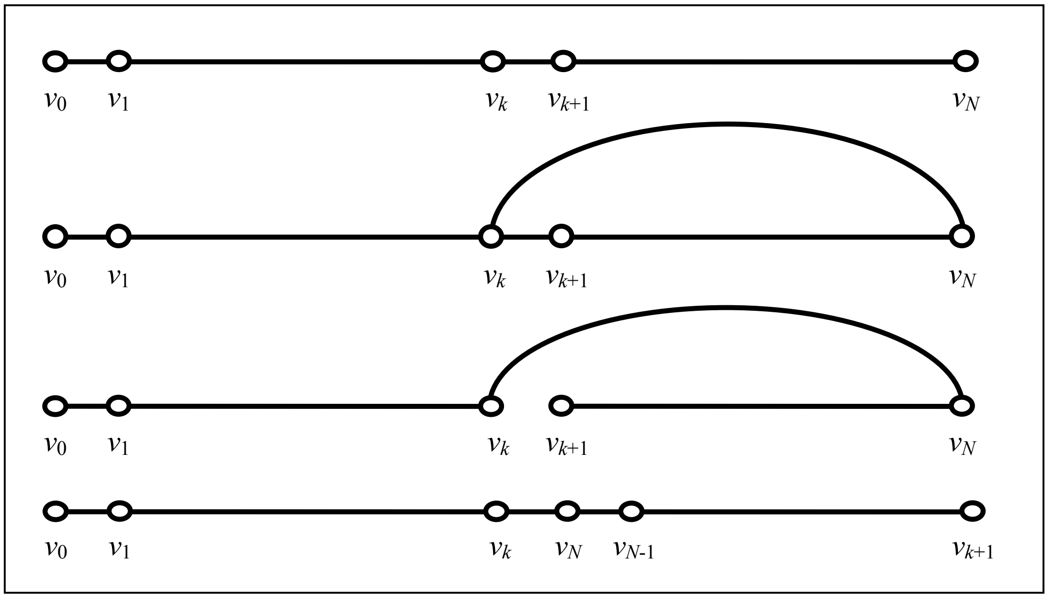


Figure 4.1. Illustration of LK algorithm

There is a restriction that the total edge cost of the spanning tree plus edge (*N,k*) in step 3 above must be less than the cost of the best tour found so far.

In this study the move was derived from the original one of LK algorithm, but it differs in such way that we omit this restriction to allow more moves to be considered and we defined *k* as 1<*k*<(*N*-1) since in this study we set the node that has largest demand as the first node to be visited from the depot. The LK algorithm can be illustrated in Figure 4.1.

4.2.3 Tabu Moves

To avoid going back to solutions that have been visited in the last few iterations (cycling), solutions that were recently examined are forbidden or declared tabu for a certain number of iterations. Tabu list is a list of moves that are currently tabu. Thus the tabu list is an ordered queue containing forbidden moves; whenever a move is made, it is inserted to the end of the tabu list and the first element from the list is removed. For the first iteration, set tabu tenure for all edges is 0. Two types of tabu duration (or tabu list size) are implemented in this study, the first is static tabu list size, which is 3, and dynamic tabu list size. In dynamic tabu list size, if an edge is moved at iteration v, its addition or deletion is tabu until iteration $v + \theta$, where θ is randomly selected in the interval [N-3, N]. The idea of using random tabu list size was introduced by Taillard in 1991 in the context of the quadratic assignment problem and was shown to reduce significantly the probability of cycling.

The best admissible move is then chosen as the highest evaluation move in the neighbourhood of the current solution in terms of the objective function value and the tabu restrictions. In some studies the whole neighbourhood is explored and the best non-tabu move is selected; however, in this study we also considering the feasible tabu improving move. An improving move is not accepted if it is forbidden in the tabu list unless it satisfies the aspiration criterion. Intensification strategies can be applied to accentuate the search in a promising region of the solution space, so that the moves to the local optimum are intensified. Diversification, on the other hand, can be used to broaden the search into less explored regions by forcing the moves out of the local optimum.

The concept of tabu move is:

1. It is tabu to add an edge which is on the deleted-list.
2. It is tabu to delete an edge which is on the added-list.

4.2.4 Aspiration Criterion

The aspiration criterion is a measure solely designed to override the tabu status of a move if this move leads to a solution better than the best found by the search so far. As a rule, the search process moves at each iteration from the current solution to the best non tabu solution in neighbourhood solutions. However, a tabu solution may be selected if the value of objective function decreased upon the best value of a solution (aspiration criterion) so far encountered. We also used aspiration by default that were:

- free the "least tabu tenure" move
- if there are more than one "least tabu tenure" moves that have same tabu tenure values, free the one with smallest objective value

4.2.5 Stopping Criteria

In our implementation, the number of iteration is equal to 10.

The Tabu Search Algorithm described above can be summarized in Figure 4.2.

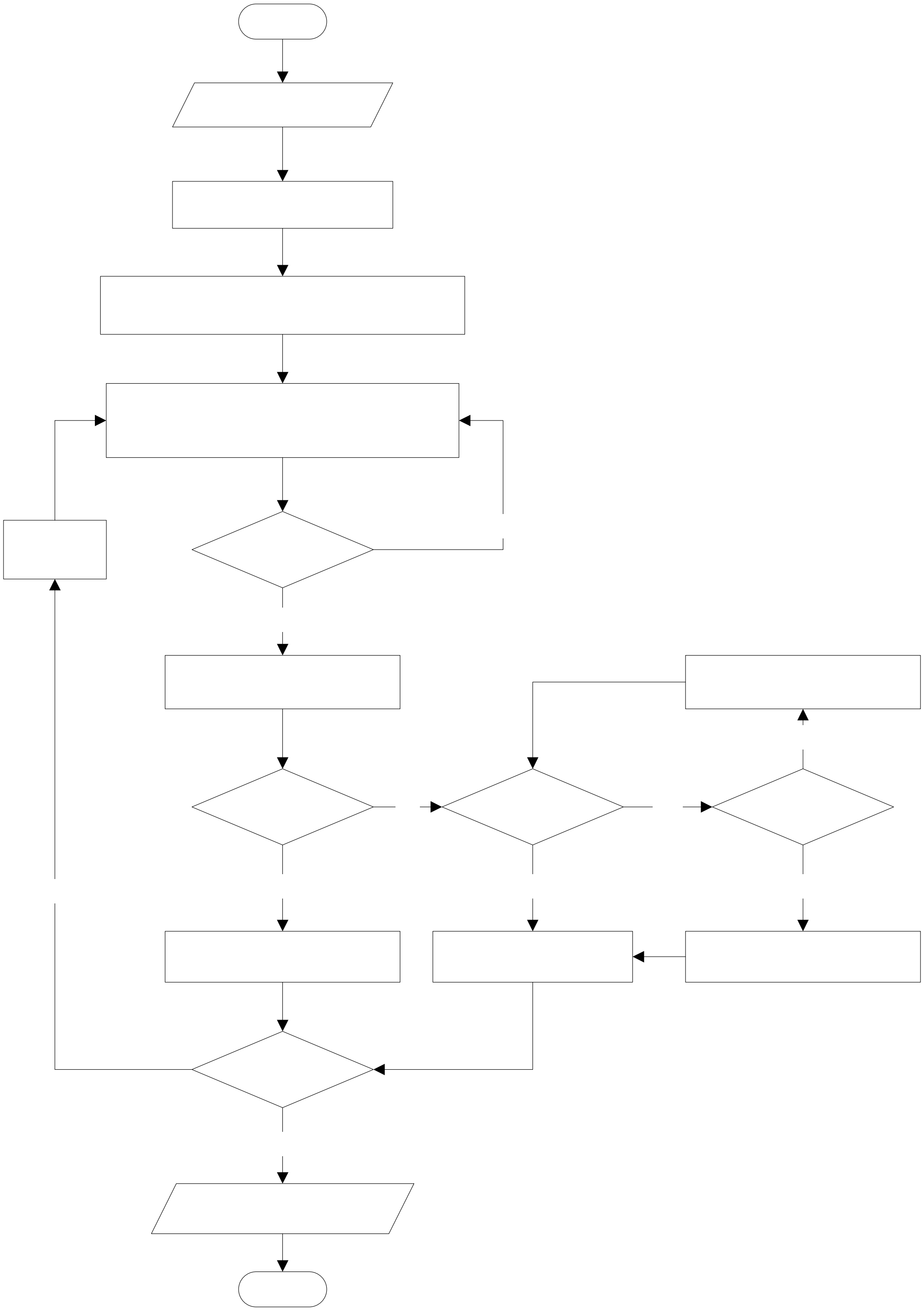


Figure 4.2. Tabu Search Algorithm for Single VRPSD

Set itera

C

C

(Move:

Find

4.3 Illustration of the Tabu Search for Single VRPSD

Consider Example 3.2 at sub section 3.11. Let *f* = 1.1 thus *Q* is equal to 11 (based on equation 3.1).

STEP 1 (Initialization). Firstly we generate a tour from Nearest Neighbour heuristic. It follows that node 2 is the node that has largest demand. Nearest Neighbour results a deterministic tour as:

0 → 2 → 4 → 5 → 3 → 1 → 0

The second heuristic that is Farthest Insertion also results the same tour. In this case, we randomly chose one of them. A priori route is depicted in Figure 4.4 with the VRPSD objective function is 269. Set this value as aspiration cost.

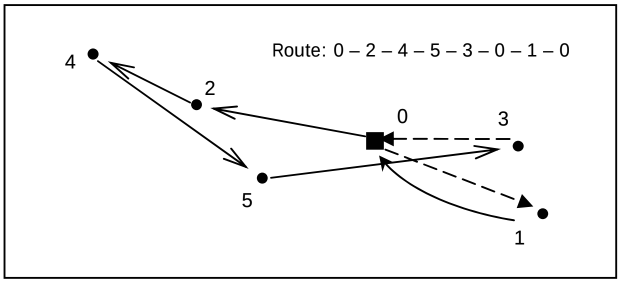


Figure 4.3. Initial Solution

STEP 2 (Neighbourhood search). Set iteration v = *v* +1, tabu tenure = 0, aspiration cost = 269. Consider all candidate moves in the neighbourhood by using 2-opt and modified LK. Evaluate it to determine the best admissible move.

STEP 3 (Coefficient update). Update the tabu list.

STEP 4 (Termination). Repeat the search process until stopping criteria is met.

Detail results on move, aspiration criteria, and tabu list are presented in Figure 4.4 and 4.5 for 2-opt and Figure 4.6 and 4.7 for modified LK. This section only considered static tabu list size. The comparison between result from static and dynamic tabu list size were discussed at Section 4.4.

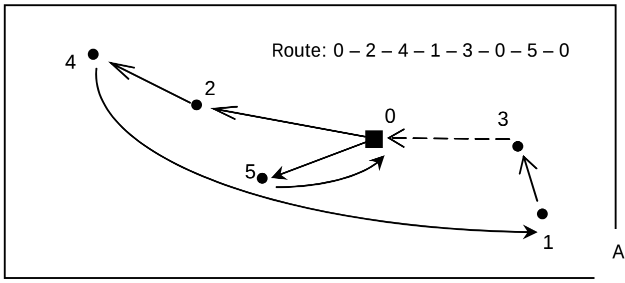
Iteration 1

Delete	Add	Route	VRPSD cost
24 & 53	25 & 43	0 2 5 4 3 1 0	315
24 & 31	23 & 41	0 2 3 5 4 1 0	330
24 & 10	21 & 40	0 2 1 3 5 4 0	292
45 & 31	43 & 51	0 2 4 3 5 1 0	289
45 & 10	41 & 50	**0 2 4 1 3 5 0**	**248**
53 & 01	51 & 30	0 2 4 5 1 3 0	258

Aspiration cost = 248

Current solution : 0 2 4 1 3 0 5 0

Tabu list : 45 (3) 10 (3)
41 (3) 50 (3)



Aspiration cost = 247

Current solution : 0 2 4 3 1 0 5 0

Tabu list : 45(0) 10(2)
41(2) 50(2)
43(3) 15(3)
35(3)

Iteration 2

Delete	Add	Route	VRPSD cost
24 & 13	21 & 43	0 2 1 4 3 5 0	356
24 & 35	23 & 45*	0 2 3 1 4 5 0	309
24 & 50*	25 & 40	0 2 5 3 1 4 0	282
41* & 35	43 & 15	**0 2 4 3 1 5 0**	**247**
41 & 50*	45* & 10*	0 2 4 5 3 1 0	269
13 & 50*	15 & 30	0 2 4 1 5 3 0	279

Aspiration cost = 247

Current solution : 0 2 3 4 1 0 5 0

Tabu list : 10(1)
41(1) 50(1)
43(2) 15(2)
35(2) 24(3)
31(3) 23(3)
41(3)

Iteration 3

Delete	Add	Route	VRPSD cost
24 & 31	23 & 41	**0 2 3 4 1 5 0**	**356**
24 & 15	21 & 45	0 2 1 3 4 5 0	308
24 & 50	25 & 40	0 2 5 1 3 4 0	271
43* & 15	41 & 35	0 2 4 1 3 5 0	248
43 & 50	45 & 30	0 2 4 5 1 3 0	258
31 & 50	35 & 10	0 2 4 3 5 1 0	289

Figure 4.4. 2-opt Move Process at Iteration 1 to 3

Iteration 4

Delete	Add	Route	VRPSD cost	Σ tabu tenure
23* & 41*	24* & 31*	0 2 4 3 1 5 0	247	12
23* & 15*	21 & 35*	0 2 1 4 3 5 0	356	7
23* & 50*	25 & 30	0 2 5 1 4 3 0	318	4
34* & 15*	31* & 45*	0 2 3 1 4 5 0	309	6
34* & 50*	35* & 40	**0 2 3 5 1 4 0**	**334**	**3**
41* & 50*	45* & 10*	0 2 3 4 5 1 0	335	6

Aspiration cost = 248

Current solution : 0 2 3 5 1 0 4 0

Tabu list :		
	40 (3)	34 (3)
	24 (2)	31 (2)
	23 (2)	41 (2)
	35 (1)	43 (1)
	15 (1)	

The process is repeated until iteration = 10

Iteration 5 **: 0 2 3 1 5 4 0 [292]**

Iteration 6 **: 0 2 4 1 5 3 0 [279]**

Iteration 7 **: 0 2 4 5 1 3 0 [258]**

Iteration 8 **: 0 2 3 5 1 4 0 [334]**

Iteration 9 **: 0 2 3 1 5 4 0 [271]**

Iteration 10

Delete	Add	Route	VRPSD cost	Σ tabu tenure
23* & 15	21 & 35*	0 2 1 3 5 4 0	292	5
23* & 54	25 & 34	**0 2 5 1 3 4 0**	**271**	**2**
23* & 40*	24* & 30	0 2 4 1 5 3 0	279	6
31* & 54	35* & 14*	0 2 3 5 1 4 0	334	9
31* & 40*	34 & 10	0 2 3 4 5 1 0	335	5
15 & 40*	14* & 50	0 2 3 1 4 5 0	309	5

Aspiration cost = 247

Current solution : 0 2 5 1 3 0 4 0

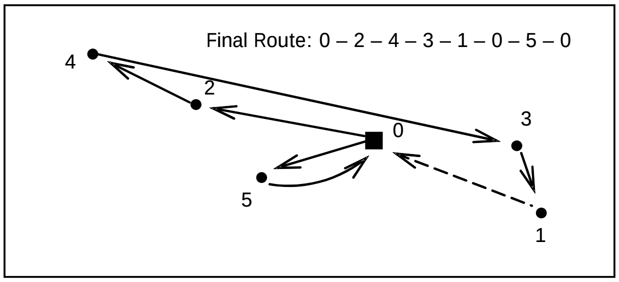


Figure 4.5. 2-opt Move Process at Iteration 4 to 10

The results of 2-opt are presented in Figure 4.4 and 4.5. Tabu Search starts from aspiration cost of 269 yields by initial solution. After that all neighbourhood that can be reached from initial solution is examined in iteration 1. Among the neighbourhood, route 0 2 4 1 3 5 0 gives the least cost (248) and it is less than aspiration cost, thus we accept it as current solution and override the aspiration cost. From the current solution, we continue to iteration 2 and examine again the neighbourhood and find that route 0 2 4 3 1 5 0 produces least cost (247). Although this move is tabu, we accept this as new current solution and override the aspiration cost since this cost is less than aspiration cost, the aspiration cost becomes 247 and we freeing the tabu status of this move.

At iteration 3, the least cost (248) is gained by route 0 2 4 1 3 5 0. Since this cost is not lower than aspiration cost and this move is tabu, thus we go to the 2nd best. This process repeat until non tabu move is found. Finally the process stop at route 0 2 3 4 1 5 0 produced from drop 24 and 31 and add 23 and 41 which are non tabu moves, so we accept it as current solution but we do not need to update the aspiration cost. When it comes to iteration 4, the entire move is tabu. So one of the neighbourhood had to be free with aspiration by default (freeing the least tabu tenure) to be the current solution for next iteration. The least tabu tenure is 2, provided by 0 2 5 1 3 4 0. These search process is repeated until stopping criteria (maximum number of iteration is equal to 10) is met. Note that at iteration 8, the current solution is cycling back to previous current solution at iteration 4, and the next iteration (iteration = 9) still produces the same route as obtained in iteration 5.

The TS results from Modified LK can be seen in Figure 4.6 and 4.7 below.

Iteration 1

Delete	Add	Route	VRPSD cost
21	24	0 2 1 3 5 4 0	303
41	45	**0 2 4 1 3 5 0**	**248**
51	53	0 2 4 5 1 3 0	258

Aspiration cost = 248

Current solution : 0 2 4 1 3 5 0

Tabu list : 41(3) and 45(3)

Iteration 2

Delete	Add	Route	VRPSD cost
24	25	0 2 5 3 1 4 0	282
41*	45*	0 2 4 5 3 1 0	269
13	15	**0 2 4 1 5 3 0**	**279**

Aspiration cost = 248

Current solution : 0 2 4 1 5 3 0

Tabu list : 41(2) 45(2)
13(3) 15(3)

Iteration 3

Delete	Add	Route	VRPSD cost
24	23	**0 2 3 5 1 4 0**	**334**
41*	43	0 2 4 3 5 1 0	289
15*	13*	0 2 4 1 3 5 0	248

Aspiration cost = 248

Current solution : 0 2 3 5 1 4 0

Tabu list : 41(1) 45(1)
13(2) 15(2)
24(3) 23(3)

Iteration 4

Delete	Add	Route	VRPSD cost
23*	24*	0 2 4 1 5 3 0	279
35	34	0 2 3 4 1 5 0	356
51	54	**0 2 3 5 4 1 0**	**330**

Aspiration cost = 248

Current solution : 0 2 3 5 4 1 0

Tabu list : 13(1) 15(1)
24(2) 23(2)
51(3) 54(3)

Iteration 5

Delete	Add	Route	VRPSD cost
23*	21	0 2 1 4 5 3 0	336
35	31	**0 2 3 1 4 5 0**	**309**
54*	51*	0 2 3 5 1 4 0	334

Aspiration cost = 248

Current solution : 0 2 3 1 4 0

Tabu list : 24(1) 23(1)
51(2) 54(2)
35(3) 31(3)

Figure 4.6 . Modified LK Move Process at iteration 1 to 5

Iteration 6

Delete	Add	Route	VRPSD cost
23*	25	0 2 5 4 1 3 0	315
31*	35*	0 2 3 5 4 1 0	330
14	15	**0 2 3 1 5 4 0**	**292**

Aspiration cost = 248

Current solution : 0 2 3 1 4 5 0

Tabu list : 51(1) 54(1)
35(2) 31(2)
14(3) 15(3)

Iteration 7

Delete	Add	Route	VRPSD cost
23	24	**0 2 4 5 1 3 0**	**258**
31*	34	0 2 3 4 5 1 0	335
15*	14*	0 2 3 1 4 5 0	309

Aspiration cost = 248

Current solution : 0 2 3 1 5 4 0

Tabu list : 35(1) 31(1)
14(2) 15(2)
23(3) 24(3)

Iteration 8

Delete	Add	Route	VRPSD cost
24*	23*	0 2 3 1 5 4 0	292
45	43	**0 2 4 3 1 5 0**	**247**
51	53	0 2 4 5 3 1 0	269

Aspiration cost = 247

Current solution : 0 2 4 5 1 3 0

Tabu list : 14(1) 15(1)
23(2) 24(2)
45(3) 43(3)

Iteration 9

Delete	Add	Route	VRPSD cost
24*	25	0 2 5 1 3 4 0	271
43*	45*	0 2 4 5 1 3 0	258
31	35	**0 2 4 3 5 1 0**	**289**

Aspiration cost = 247

Current solution : 0 2 4 3 1 5 0

Tabu list : 23(1) 24(1)
45(2) 43(2)
31(3) 35(3)

Iteration 10

Delete	Add	Σ tabu tenure	Route	VRPSD cost
24*	21	**1**	**0 2 1 5 3 4 0**	**333**
43*	41	2	0 2 4 1 5 3 0	279
35*	31*	6	0 2 4 3 1 5 0	247

Aspiration cost = 247

Current solution : 0 2 4 3 5 1 0

Tabu list : 45(1) 43(1)
31(2) 35(2)
24(3) 21(3)

Figure 4.7 . Modified LK Move Process at iteration 6 to 10

4.4 Discussion

Figure 4.4 to 4.7 present the neighbourhood structure of 2-opt and Modified LK. Compared to 2-opt, LK has several advantages:

- easier to implement
- less number of move to be examined at each iteration

but it has disadvantage since it need longer iteration than 2-opt to reach the global optimum. 2-opt reaches the best solution at iteration 3 whereas Modified LK reaches the same best cost value as 2-opt at iteration 8. This condition is depicted in Figure 4.8.

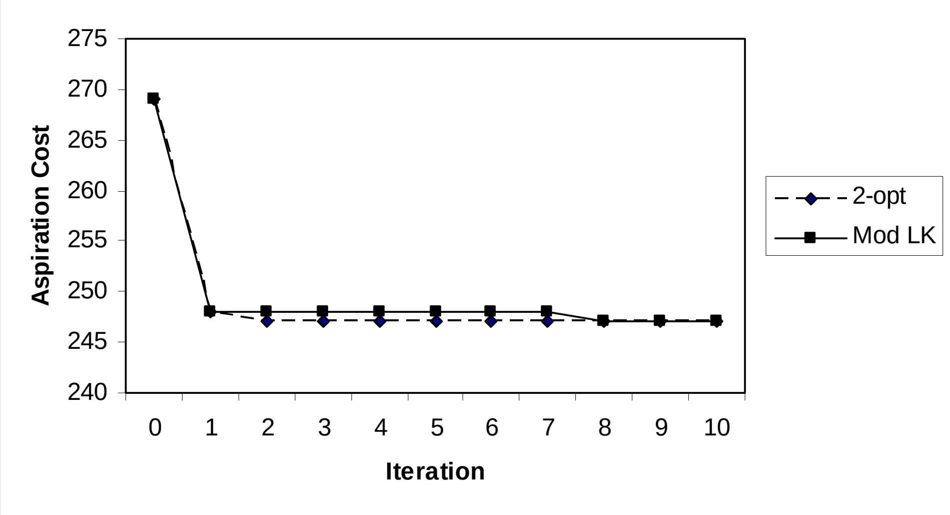


Figure 4.8. Aspiration Cost versus Iteration

As shown in Figure 4.9, TS allows the search to accept non improving move as long as it is not tabu, and at the other side it also allows tabu move as long it can give an improvement to the aspiration cost. The results obtained by 2-opt and Modified LK are compared. The 2-opt and Modified LK gave the same optimal value of 247.

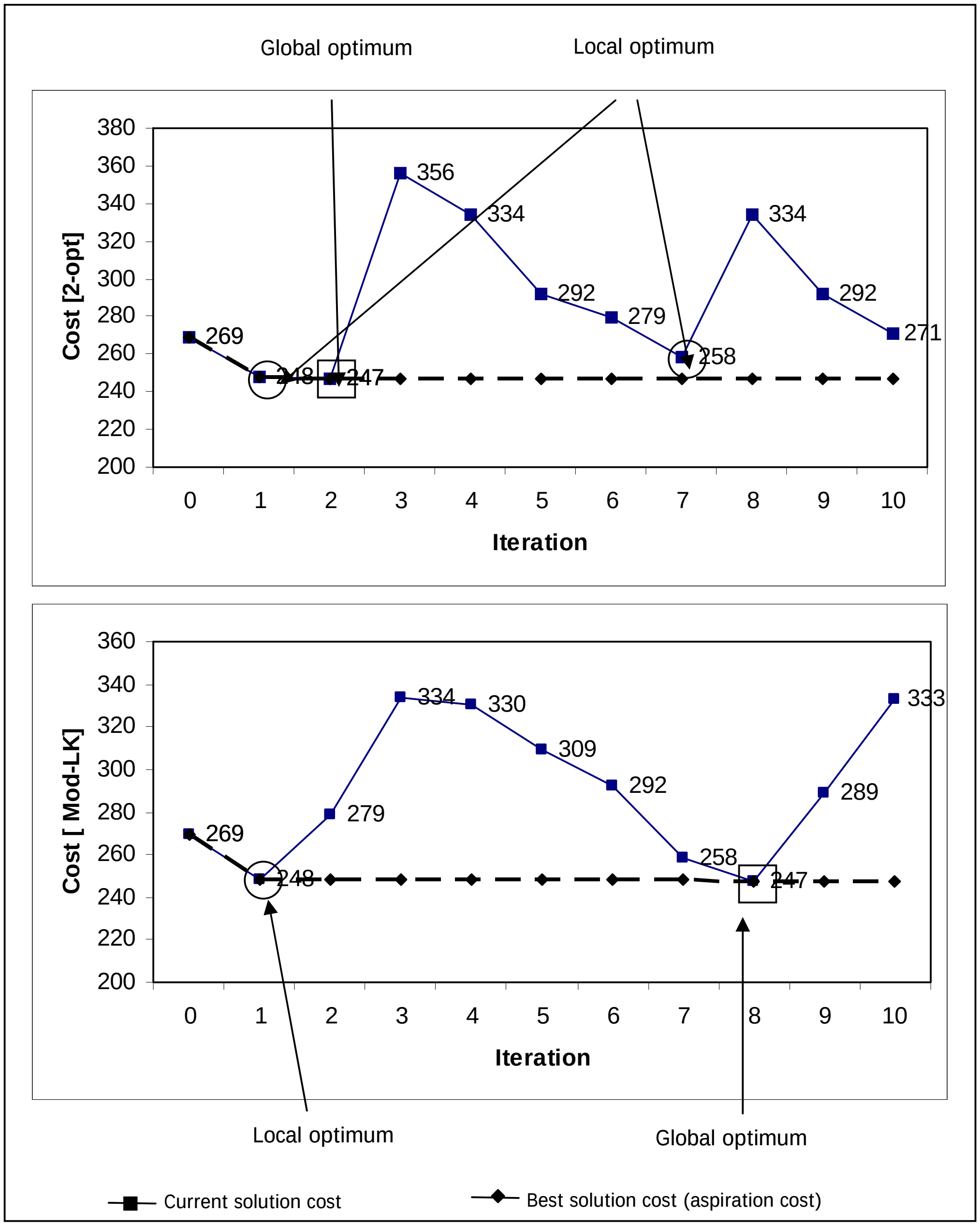
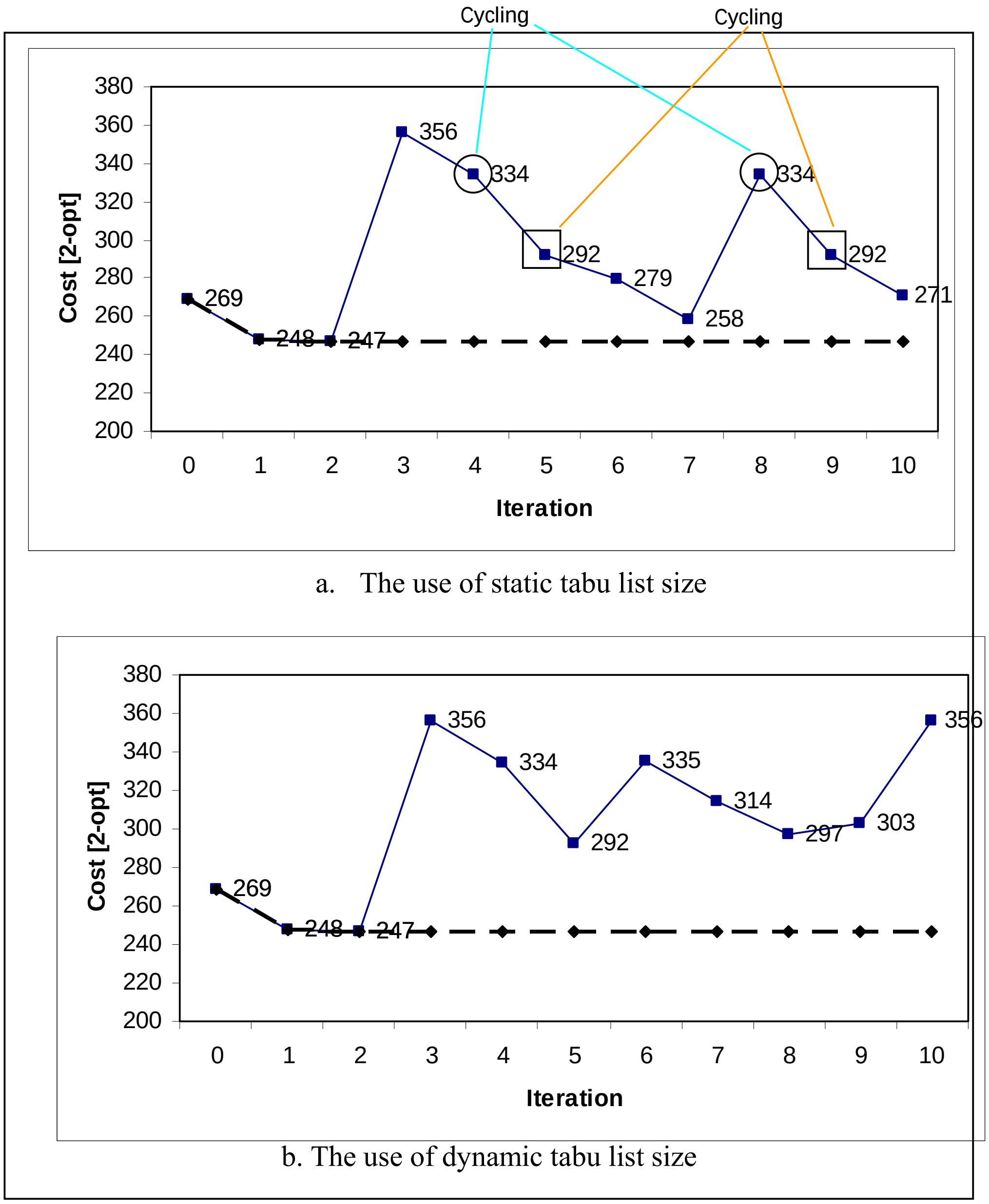


Figure 4.9. Cost versus Iteration

In this section we have explain the result of the use of two different moves that were 2-opt and modified Lin and Kernighan. From the previous discussion, Tabu Search is proven to be able to yield a good quality solution. Further, we compared the use of static and dynamic tabu list size, and the effect to the search process was investigated.



a. The use of static tabu list size

b. The use of dynamic tabu list size

Figure 4.10. The comparison between static and dynamic tabu list size

Although the best result from both types of tabu list size was the same, but it was quite different in the search process values. Actually tabu list was used to prevent the search from cycling, but static tabu list size shows weakness since we can still find solutions that have been reached from a few last iterations. By using dynamic tabu list size, the probability of cycling can be reduced significantly as depicted in Figure 4.10. Please note that in Figure 4.10b, the objective function at iteration 10 was the same like the one at iteration 3. It does not mean that cycling was occurred since the routes produced from those two iterations were different but they produce the same objective value.

CHAPTER 5

GENETIC ALGORITHM FOR SOLVING SINGLE VRPSD

5.1 Introduction

Genetic Algorithm (GA) is a search method based on principles of natural selection and genetics. Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can start to evolve solutions to the search problem using selection, recombination, and mutation. Limited number of research using GA in VRPSD has opened a new direction to develop GA for solving VRPSD quickly and reliably. We give brief description of GA that was implemented, the experimental results and the discussion.

5.2 Genetic Algorithm for Single VRPSD

5.2.1 Chromosome Representation

In developing the algorithm, the permutation representation or the path representation or order representation is used since the typical approach using binary strings will simply make coding more difficult. Order representation is perhaps the most natural and useful representation of a VRP tour, where customers are listed in the order in which they are visited. A chromosome represents a route and a gene represents a customer and the values of genes are called alleles. The search space for this representation is the set of permutations of the customers; every chromosome is a string of numbers that represent a position in a sequence.

5.2.2 Initialization

Usually the initial population of candidate solutions is generated randomly across the search space. However, other information can be easily incorporated to yield better results. In this study, the inclusion of Nearest Neighbour and Farthest Insertion to the initial population was implemented to give the GA a good starting point. The population is an array P of N (population size) chromosomes. Each chromosome P_k is initialized as a permutation of customers. Clones (identical solutions) are forbidden in P to ensure a better dispersal of solutions and to diminish the risk of premature convergence.

The population size is one of the important factors affecting the performance of genetic algorithm. Small population size might lead to premature convergence. On the other hand, large population size leads to unnecessary expenditure of valuable computational time. The population size in this study was the multiplication of number of customers, that was two times the number of customers, in order to remain diversify the population.

5.2.3 Evaluation

Once the population is initialized or an offspring population is created, the fitness values of candidate solutions are evaluated. The fitness value is the function of VRPSD objective function.

5.2.4 Roulette Wheel Selection with elitism

This research employs the combination of two types of GA selection techniques, namely roulette wheel and elitism. The roulette wheel selection works by selecting the chromosome by looking at their proportional fitness rank. This is where the evolution concept survival of the fittest comes into plays. Some researchers found that this technique will lead to only the best chromosome been selected in the population. It because the fittest chromosome rank is bigger compared to the less fit chromosome and in probability of course chromosome with the highest rate will have a big chance to be selected, while the elitism technique is a simple selection operator, which reserved the best found chromosome in the current population to rebirth for the next generation.

Mitchell (1996) said that elitism could increase rapidly the performance of genetic algorithm, because it prevents losing the best-found solution. When creating a new generation using the reproduction operator, we could have a chance of losing the best-found chromosome in the current population. This is where elitism plays their role in preventing the lost of this chromosome. In this example, after the calculation for each selection chromosome probability has been done, the elitism operator will automatically reserved the chromosome that produce the lowest expected cost.

5.2.5 Order Crossover (OX)

In Oliver et al. (1987), Partially Match Crossover (PMX), Cycle Crossover (CX) and Order Crossover (OX) were applied to the 30-city problem of Hopfield and Tank. They found that the best tour generated with OX was 11% shorter than the best PMX tour, and 15% shorter than the best CX tour. In a later study by Starkweather et al. (1991), six different crossover operators were tested on the problem of Hopfield and Tank. Thirty different runs were performed with each operator. In this experiment, OX found the optimum 25 times (out of 30), while PMX found the optimum only once, and CX never found the optimum.

Thus in this study, OX was employed in generation of offspring. OX was first proposed by Oliver et al. in 1987. This crossover operator extends the modified crossover of Davis by allowing two cut points to be randomly chosen on the parent chromosomes. In order to create an offspring, the string between the two cut points in the first parent is first copied to the offspring. Then, the remaining positions are filled by considering the sequence of cities in the second parent, starting after the second cut point (when the end of the chromosome is reached, the sequence continues at position 1).

In Figure 5.1, the substring 564 in parent 1 is first copied to the offspring (step 1). Then, the remaining positions are filled one by one after the second cut point, by considering the corresponding sequence of customers in parent 2, namely 57814236 (step 2). Hence, customer 5 is first considered to occupy position 6, but it is discarded because it is already included in the offspring. Customer 7 is the next customer to be considered, and it is inserted at position 6. Then, customerity 8 is inserted a t position 7, city 1 is inserted at position 8, city 4 is discarded, city 2 is inserted at position 1, city 3 is inserted at position 2, and city 6 is discarded.

parent 1 : **1 2 | 5 6 4 | 3 8 7**
parent 2 : 1 4 | 2 3 6 | 5 7 8

offspring: (step 1) : - - **5 6 4** - - -
(step 2) : 2 3 **5 6 4** 7 8 1

Figure 5.1**.** The Order Crossover

Clearly, OX tries to preserve the relative order of the cities in parent 2, rather than their absolute position. In Figure 11, the offspring does not preserve the position of any city in parent 2. However, city 7 still appears before city 8, and city 2 before city 3 in the resulting offspring.

5.2.6 Mutation

Mutation alters one or more genes (positions) of a selected chromosome (solution) to reintroduce lost genetic material and introduce some extra variability into the population. Mutation operators are aimed at randomly generating new permutations of the cities. As opposed to the classical mutation operator, which introduces small perturbations into the chromosome, the permutation operators for the VRP and or TSP often greatly modify the original tour. In this study, we implement two (2) mutation methods that were swap mutation and displacement mutation. In swap mutation, two customer locations are swapped, and their positions are exchanged. This mutation operator is the closest in philosophy to the original mutation operator, because it only slightly modifies the original tour. For example, choose two random positions, i.e. position 2 and 7 and swap entries from tour

7,**4**,0,3,2,1,**5**,6

and the tour becomes

7,**5**,0,3,2,1,**4**,6

In displacement mutation, firstly we select two random points, grab the chunk between them and move it somewhere else, for example:

0,1,2,**3**,4,**5**,6,7 becomes 0,**3,4,5,**1,2,6,7.

5.2.7 Stopping criterion

In our implementation, the GA procedure is repeated until the maximum number of generation.

The Genetic Algorithm is depicted in Figure 5.2.

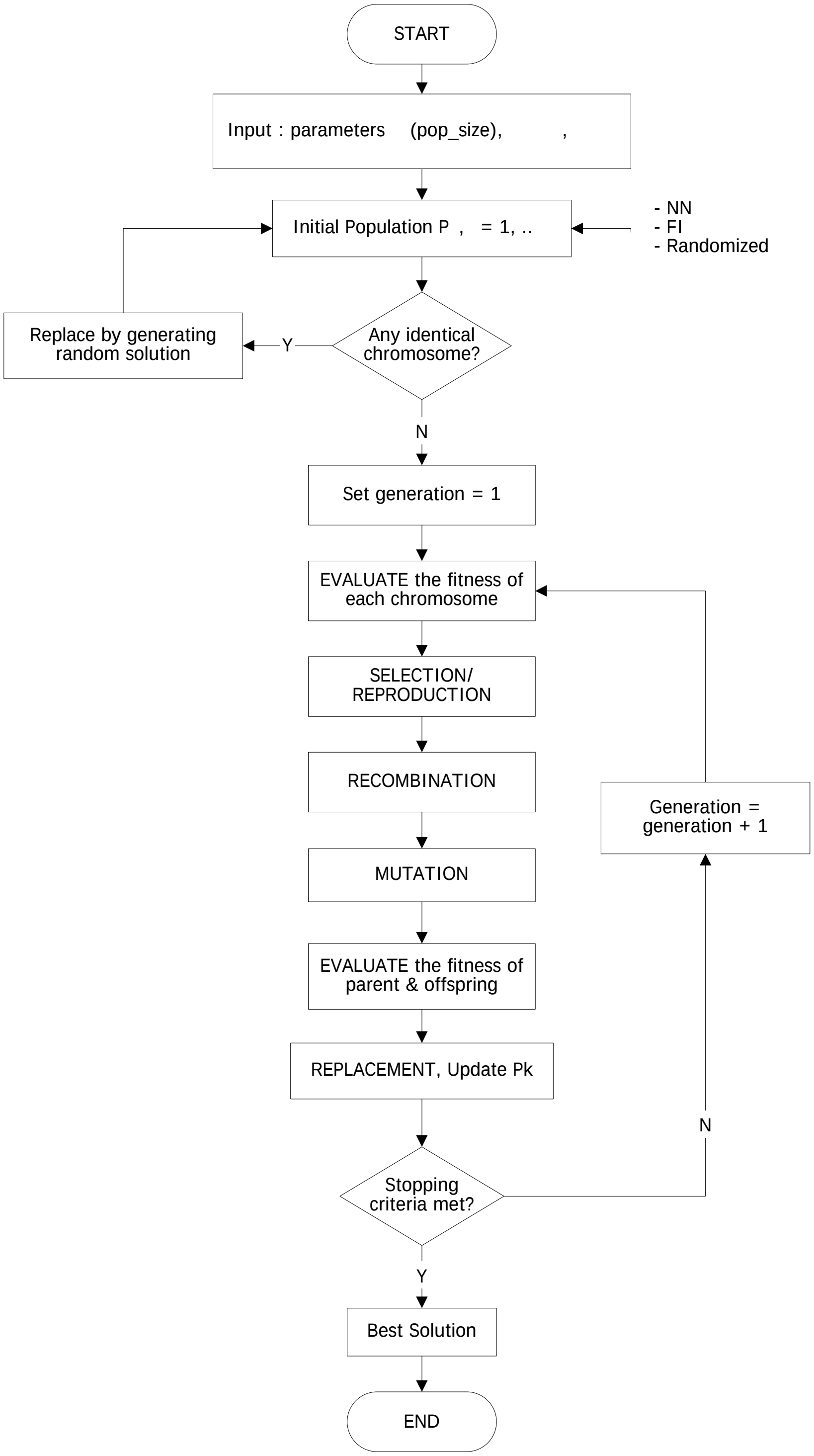

START
Input : parameters (pop_size), ,
Initial Population P , = 1, ..
- NN
- FI
- Randomized
Any identical chromosome?
Y
Replace by generating random solution
N
Set generation = 1
EVALUATE the fitness of each chromosome
SELECTION/ REPRODUCTION
RECOMBINATION
MUTATION
EVALUATE the fitness of parent & offspring
REPLACEMENT, Update Pk
Stopping criteria met?
N
Generation = generation + 1
Y
Best Solution
END

Figure 5.2. Structure of Genetic Algorithm

5.3 Illustration of Genetic Algorithm

In this study, the GA parameters were set at the following values:

- population size N = 10
- probability of crossover = 0.75
- probability of mutation = 0.05
- number of generation = 10

The initial population generated by Nearest Neighbour and Farthest Insertion heuristics and randomly generated candidate solutions were listed in Table 5.1. Chromosome that identical to the other chromosomes was replaced by generating candidate solution randomly until there are no clones.

Table 5.1. Population in the initial Generation (Generation = 0)

Chromosome No	Generated by	Route	Route without clones
1	FI	0 2 4 5 3 1 0	0 2 4 5 3 1 0
2	NN	0 2 4 5 3 1 0	0 2 1 5 3 4 0
3	Random	0 2 1 3 5 4 0	0 2 1 3 5 4 0
4	Random	0 2 3 1 5 4 0	0 2 3 1 5 4 0
5	Random	0 2 4 1 3 5 0	0 2 4 1 3 5 0
6	Random	0 2 4 5 3 1 0	0 2 1 4 5 3 0
7	Random	0 2 3 5 1 4 0	0 2 3 5 1 4 0
8	Random	0 2 3 5 4 1 0	0 2 3 5 4 1 0
9	Random	0 2 5 1 4 3 0	0 2 5 1 4 3 0
10	Random	0 2 5 1 3 4 0	0 2 5 1 3 4 0

After the population was built, the evaluation and selection took place. VRPSD objective function is a evaluation measure for discriminating good solutions and bad solutions. The smaller the VRPSD objective function, the better the solution. In roulette wheel selection, each individual in the population is assigned a roulette wheel slot sized in proportion to its fitness. That is, good solutions have a larger slot size than the less fit solution. Set VRPSD objective function to be *f*. Thus, we use 1-(fi/sum (fi)) as fitness function rather than VRPSD objective function itself. The larger the fitness value, the higher probability of the chromosome to be chosen as parent. It is possible for a chromosome to be selected multiple times. The evaluation and selection process can be seen in Table 5.2.

Table 5.2. Evaluation and Selection in the 1st generation

Chromosome number	Population	VRPSD objective function (fi)	1-(fi/sum(fi))	cumulative	Random number for selection	Chrom. selected
(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**
1	0 2 4 5 3 1 0	269	0.911338	0.911338	5.1548	6
2	0 2 1 5 3 4 0	333	0.890244	1.801582	4.12938	5
3	0 2 1 3 5 4 0	303	0.900132	2.701714	4.54941	6
4	0 2 3 1 5 4 0	292	0.903757	3.605471	6.3577	8
5	0 2 4 1 3 5 0	248	0.918260	4.523731	5.81892	7
6	0 2 1 4 5 3 0	336	0.889255	5.412986	2.41511	3
7	0 2 3 5 1 4 0	334	0.889914	6.3029	6.78034	8
8	0 2 3 5 4 1 0	330	0.891233	7.194133	4.85327	6
9	0 2 5 1 4 3 0	318	0.895188	8.089321	2.07099	3
10	0 2 5 1 3 4 0	271	0.910679	9	3.14572	4
SUM		3034				
AVERAGE		303.4				

Once the evaluation and selection were done, crossover and mutation were applied (as shown in Table 5.3) in order to create new, possibly better solution. For each chromosome, a random number between 0 and 1 is generated. If the random number is less than the crossover probability input parameter (=0.75), the chromosome is picked for crossover. Each time two chromosomes have been picked up, OX crossover is applied and the population is listed. There are three pairs of chromosome to be recombined and the population size became 13: 10 parents and 3 offspring. Each chromosome in the population listed after crossover operation is now given the chance to mutate by generating random number between 0 and 1. If the random number is less than the mutation rate, the chromosome is then picked for mutation. The offspring then replaced the worst parents until the population size is equal to 10. If none of the offspring is better than the parents, than the population just remains the same. These procedures were repeated until the number of generation is equal to 10.

Table 5.3. Crossover and Swap Mutation Process in the 1st generation

MATING POOL	Random No for crossover	Parent and Offspring	Random No for mutation	After crossover & mutation	Cost	Cost Rank	New Population
(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**
0 2 1 4 5 3 0	0.853103	0 2 **1 4** 5 3 0	**0.04655**	0 2 4 1 5 3 0	279	3	0 2 4 1 5 3 0
0 2 4 1 3 5 0	**0.125565**	0 2 4 1 3 5 0	0.60446	0 2 4 1 3 5 0	248	1	0 2 4 1 3 5 0
0 2 1 4 5 3 0	**0.391876**	0 2 1 4 5 3 0	0.92949	0 2 1 4 5 3 0	336		0 2 3 5 4 1 0
0 2 3 5 4 1 0	**0.600514**	0 2 3 5 4 1 0	0.31041	0 2 3 5 4 1 0	330	9	0 2 1 3 5 4 0
0 2 3 5 1 4 0	**0.340618**	0 2 3 5 1 4 0	0.88872	0 2 3 5 1 4 0	334		0 2 3 5 4 1 0
0 2 1 3 5 4 0	**0.524349**	0 2 1 3 5 4 0	0.71238	0 2 1 3 5 4 0	303	7	0 2 1 3 5 4 0
~~0 2 3 5 4 1 0~~	0.979810	0 2 3 5 4 1 0	0.93372	0 2 3 5 4 1 0	330	10	0 2 3 1 5 4 0
0 2 1 4 5 3 0	0.948004	0 2 1 4 5 3 0	0.20853	0 2 1 4 5 3 0	336		0 2 5 1 3 4 0
0 2 1 3 5 4 0	0.878514	0 2 1 3 5 4 0	0.24742	0 2 1 3 5 4 0	303	6	0 2 1 5 4 3 0
~~0 2 3 1 5 4 0~~	**0.230942**	0 2 3 1 5 4 0	0.19517	0 2 3 1 5 4 0	292	4	0 2 1 3 5 4 0
		0 2 5 1 3 4 0	0.60567	0 2 5 1 3 4 0	271	2	
		0 2 1 5 4 3 0	0.16681	0 2 1 5 4 3 0	319	8	
		0 2 1 3 5 4 0	0.29362	0 2 1 3 5 4 0	303	5	
AVERAGE							297.8
BEST COST							248

From generation to generation, we may get better and better chromosomes in the population as shown in Table 5.4. The parent chromosomes have already been selected according to their fitness, so the next population (which included the parents which did not undergo crossover) is among the fittest in the population and the population was gradually, on average, increase its fitness (which mean the VRPSD objective function is decreasing). The problem is the condition when the two parents have the same allele at a given gene then OX crossover will not change that. In other words, the offspring remains the same like their parents and it caused the average of the fitness was not changed. Mutation is designed to overcome this problem in order to add diversity to the population and ensure that it is possible to explore the entire search space. But in fact, using swap mutation with the probability of 0.05 did not make any substantial change that could create a new better chromosome. So, better mutation operator is needed to explore more the search region.

Table 5.4. GA Results from Swap Mutation

Generation	0	1	2	3	4	5	6	7	8	9	10
Best	269	248	248	248	248	248	248	248	248	248	248
Average	303.4	297.8	276.9	264.1	261.8	261.8	261.8	261.8	261.8	261.8	261.8

5.4. Comparison between GA and TS

Table 5.5. GA and TS Results

Generation/ Iteration	0	1	2	3	4	5	6	7	8	9	10
Best GA	269	248	248	248	248	248	248	248	248	248	248
Best TS (2-opt)	269	248	247	247	247	247	247	247	247	247	247
Best TS (Mod-LK)	269	248	248	248	248	248	248	248	247	247	247

The GA results were compared to the TS results from previous chapter. For single run and the number of iteration is equal to 10, TS with 2-opt and Mod-LK seems to reach optimal solution, although they were different from the iteration number when they got optimal solution. The interesting thing was GA could not reach an optimal solution for the same number of iteration. It is possibly since the very simple swap mutation with probability = 0.05 used could not add diversity in the solution and could not explore more the search region. Better mutation operator was suggested for the further study.

CHAPTER 6

SOFTWARE DEVELOPMENT FOR SOLVING SINGLE VRPSD

6.1 Introduction

This chapter explains the Vehicle Routing Problem with Stochastic Demands' program operation which built using Microsoft Visual studio C++ 6.0. It is a user friendly program developed to assist the users in deciding the route to take to accomplish the task of waste collection in the least total expected cost by following the simple policy as mentioned in Chapter 2. Each of the part and its function will be described in details.

Besides, this chapter provides information about the implementation of Visual Studio C++ 6.0 and some main functions used in developing this program.

6.2 Scope and Parameter Setting of the Designed Program

This designed program has been scoped for some aspects. Problem set with maximum number of customers that can be solved by this program is 100 units of customers. Demands of customers are recorded as a range where total number of possible demands for each customer cannot exceed 100 units. Capacity of vehicle used can be determined by user up to 100 units. The scope and parameter settings of this designed program are summarized in Table 6.1 and Table 6.2 respectively.

Table 6.1. Scope of program

Maximum number of customers, *n*	100
Maximum capacity of vehicle, *Q*	100
Total possible demands for each customer	100

Table 6.2. Parameter setting for program

Penalty for route failure, *b*	2
a_0	0.9
α, ρ	0.1
β	2
Number of ants, *m* (for *n* > 10)	10
Maximum iterations (ACS)	100
Num. of consecutive non-improving solutions allowed (ACS)	50
Initial temperature (SA)	1000
Number of repetitions (SA)	20
Cooling Rate (SA)	0.98
Total of Iterations (SA)	1500

6.3 Operation of the Designed Program

In this section, the graphic user interface of the program is shown in Figure 6.1. There are three main parts on this interface.

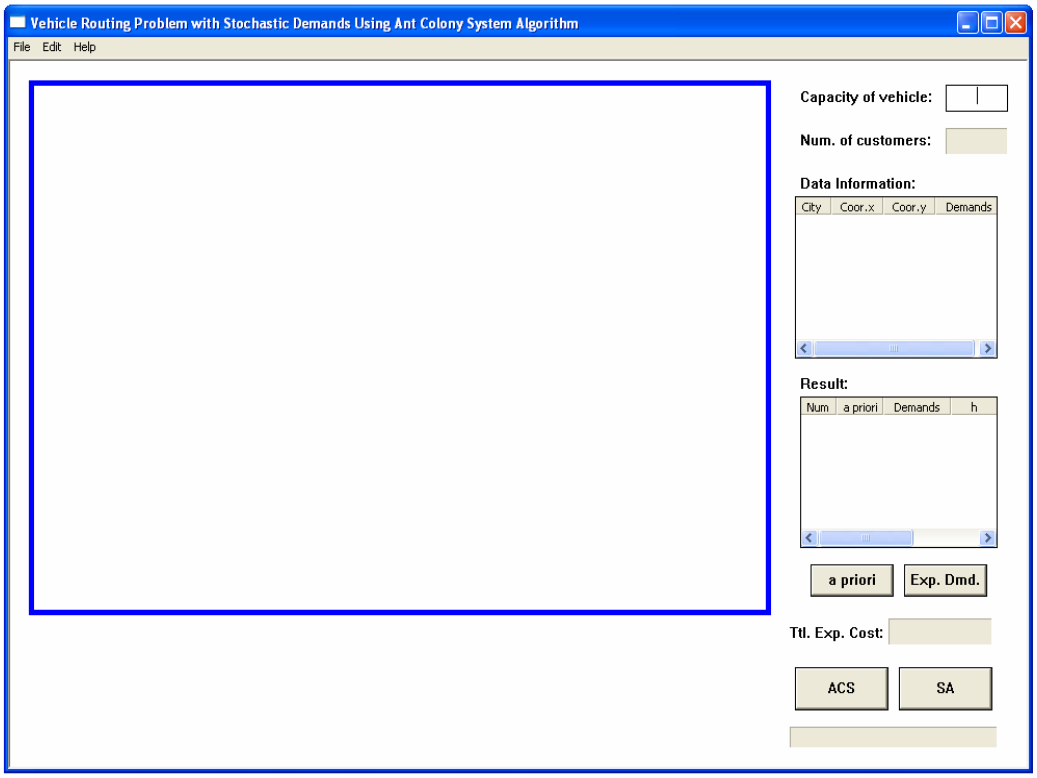
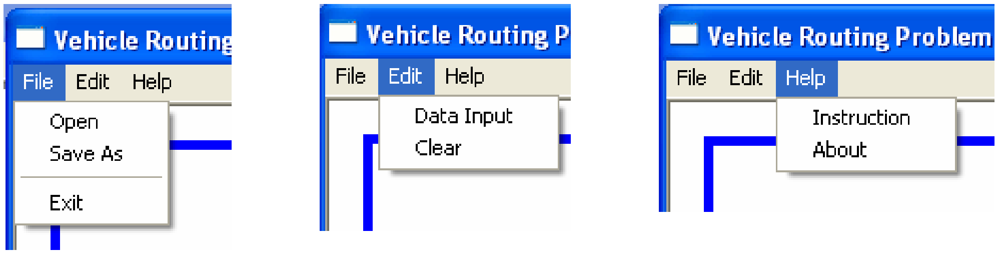


Figure 6.1. Main GUI (Graphical User Interface) of designed program

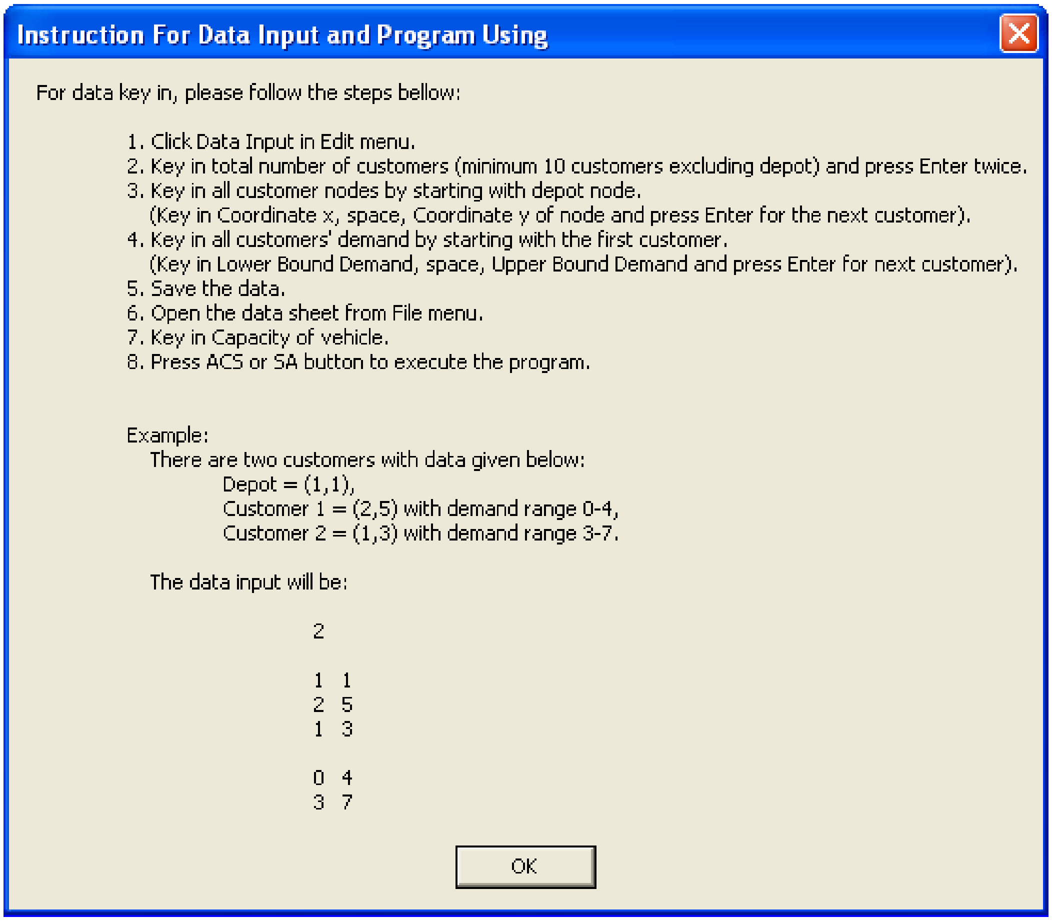
The first part is the menu bar which made up of three pull-down menus which are File, Edit and Help. A few items can be found in each of the pull-down menus of File, Edit and Help as shown in Figure 6.2(a), (b) and (c) respectively.



(a) (b) (c)

Figure 6.2. Pull-down menus and its items in menu bar

In File pull-down menu, there are common choices of Open, Save As and Exit items. While in Edit pull-down menu, there have Data Input item for user to key in the data to the notepad and Clear item for delete or clear all the information used and shown. Lastly, in Help pull-down menu, there have Instruction item which provides the procedure for data key-in and program using in detail as shown in Figure 6.3 while About item shows the information of this designed program.



For data key in, please follow the steps bellow:

1. Click Data Input in Edit menu.
2. Key in total number of customers (minimum 10 customers excluding depot) and press Enter twice.
3. Key in all customer nodes by starting with depot node.
   (Key in Coordinate x, space, Coordinate y of node and press Enter for the next customer).
4. Key in all customers' demand by starting with the first customer.
   (Key in Lower Bound Demand, space, Upper Bound Demand and press Enter for next customer).
5. Save the data.
6. Open the data sheet from File menu.
7. Key in Capacity of vehicle.
8. Press ACS or SA button to execute the program.

Example:
There are two customers with data given below:
Depot = (1,1),
Customer 1 = (2,5) with demand range 0-4,
Customer 2 = (1,3) with demand range 3-7.

The data input will be:

```
2

1  1
2  5
1  3

0  4
3  7
```

Figure 6.3. Message box in Instruction item

The main role for second part of the designed program is displaying the information of data and result. The graphical display region displays the customers' locations and route of the result. At the right hand side on the interface, there have two list controls. The upper one displays the data information which includes coordinate *x* and *y* for the customers' locations and demands for each of the customers while the lower one displays the result. Result information includes the a priori tour, demands of customers, capacity threshold value h_w, expected demand for every customer and the capacity left q in the vehicle after servicing customer w.

After user open an input file from Open item in File pull-down menu, graphical display region will displays the customers' points and total number of customer will be shown through the static box. An edit box at top of right hand side on interface is created for user to key in the maximum capacity of the vehicle used. Once the program is run, the total expected cost for the data set will be shown in the static box under the list control. All of the components in second part of program are shown in Figure 6.4.

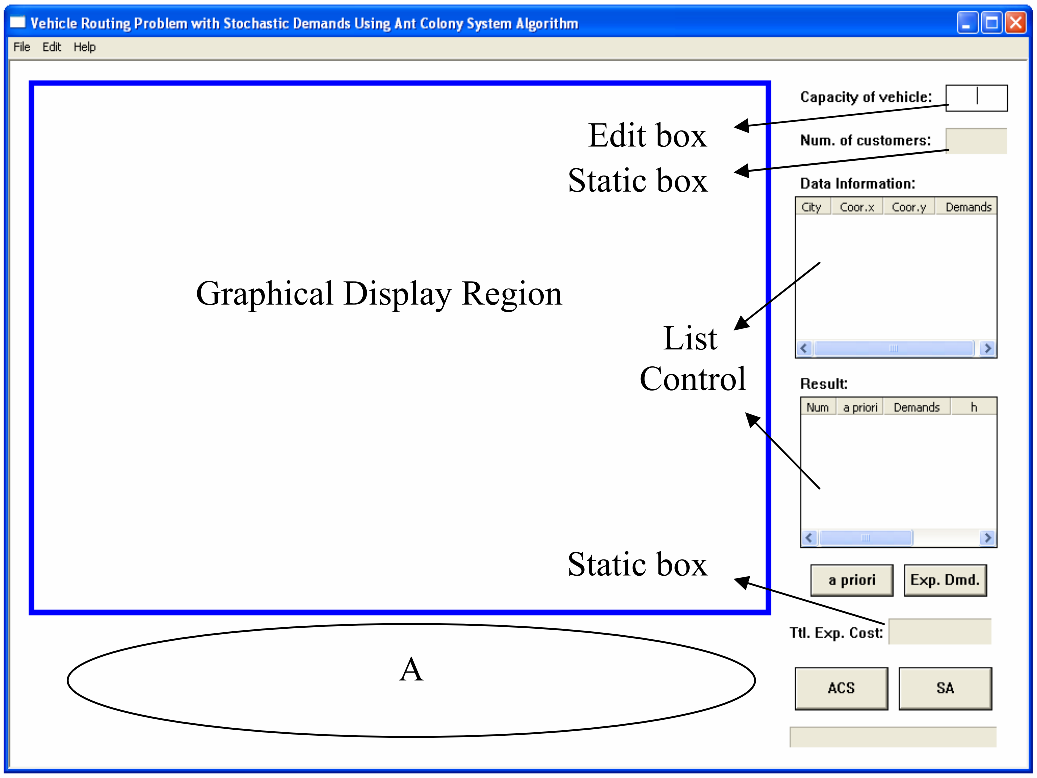


Figure 6.4. Components for data and result displaying

Lastly, third part of the program is the result computation. After open a file of data set, the user just needs to click the ACS or SA button to run the program. ACS button is used to solve the problem using Ant Colony Program while SA button solves the problem using Simulated Annealing Method. At the end of each search, the program will display the a priori tour and its total expected cost. A progress bar is created to show the progress of computation process.

Expected demand can be calculated using ‘Exp. Dmd.’ button based on the a priori tour to obtain the result for route with recourse action and its total expected cost. Route with recourse action will be shown in the region labeled with capital *A* in Figure 6.4 while its graphical tour will be displayed in the graphical display region. The a priori tour and its total expected cost of the result can be obtained again by just having a click on ‘a priori’ button.

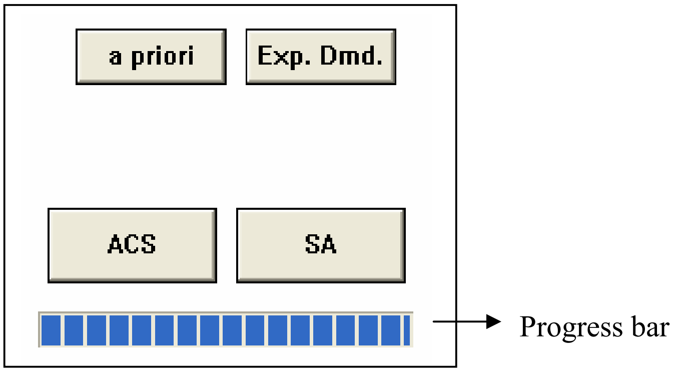


Figure 6.5 Third part of the designed program

6.4 Programming with Microsoft Visual C++ 6.0

6.4.1 Introduction

C is a computer programming language that can be used to create applications that control the computer and allow a person interact with the machine. To enhance it, another language, named C++ was developed from it by adding the concept of object-oriented programming. Object-oriented programming is a programming paradigm that uses "objects" and their interactions to design applications and computer programs.

Microsoft Visual C++ is a graphical programming environment used to create computer applications for the Microsoft Windows family of operating programs and it allows designing Windows objects and writing code to implement their behavior. To assist it, the Microsoft Foundation Class Library, or MFC, was created as an adaptation of Win32 in MS Visual Studio. The MFC is a library and can be used to manually develop programs (Waters, 1994).

6.4.2 The Visual C++ Application

When starting to do a program, an application is the first thing to be created. In MFC, this process has been resumed in a class called CWinApp (Class-For-A-Windows-Application). Based on this, one must derive a class from CWinApp to create an application.

An application by itself is an empty thing that only lets the operating program knows a program which will executed on the computer are created. It doesn't display anything on the screen. If you want to display something, the CWinApp class provides the InitApplication(), a Boolean method that must be overrided in that class. If it succeeds in creating the application, it returns TRUE. If something went wrong in creating the application, it would return FALSE. The minimum skeleton of an application would appear as follows:

```
class CExerciseApp : public CWinApp
{
public:
     virtual BOOL InitInstance();
};

BOOL CExerciseApp::InitInstance()
{
     return TRUE;
}
```

After creating the application, a global variable of the class must be declared to make it available to other parts of the program. It is usually called theApp but it can be named as anything as programmer wish.

The fundamental classes of MFC are declared in the afxwin.h header file. Therefore, this is the primary header that programmer may have to add to each one of applications. Based on this, a basic application can be created as follows:

```
#include<afxwin.h>
class CExerciseApp : public CWinApp
{
public:
     virtual BOOL InitInstance();
};

BOOL CExerciseApp::InitInstance()
{
     return TRUE;
}
CExerciseApp theApp;
```

6.4.3 Main Functions in ACS Code

A complete code is made up of some Functions with different tasks. The main functions that used in developing the designed program are summarized together with its description in Table 6.3. The complete functions and descriptions are reflected in the ACS code which can be found in Appendix A.

Table 6.3. Main functions in ACS code

Function	**Description**
`CACS()`	The constructer that initializes the variables and arrays. The function also creates the push buttons, edit boxes, static boxes and list controls.
`~CACS()`	The destructor.
`OnPaint()`	Set up the initial display.
`DrawPoint()`	Draws the points for customers' locations
`DescentMethod()`	Computes the local minima for each ant.
`AntColonySystem()`	Compute optimal solution using Ant Colony System algorithm by calling up function `DescentMethod()`.
`Swap_Customer()`	Swap any two randomly chosen customers' position.
`Simulated_Annealing()`	Compute optimal solution using Simulated Annealing algorithm by calling up the function `Swap_Customer()`.
`OnClickCalc_ACS()`	Activates the push button *ACS*.
`OnClickCalc_SA()`	Activates the push button *SA*.
`OnClickShow _A_PRIORI()`	Shows the a priori tour and its total expected cost.
`OnClickCalc_ EXPDEMAND()`	Calculates the expected demands and shows the result.
`ObjFunction()`	Computes the expected objective function value using Equation (2.1-2.4)
`Expected_Demands()`	Generate expected demands and calculate the expected objective function value
`DescentMethod()`	Compute the local minima for each ant.
`OnFileOpen()`	Read information from an existing file
`OnFileSave()`	Save the information and result to an output file.

6.5 Summary

This chapter presents the development of the ACS and SA’s Program for solving VRPSD by using Microsoft Visual Studio C++ 6.0 and goes into details about program implementation. This program was built to solve a simple vehicle routing problem with stochastic demands based on Ant Colony Program with local search. It is also includes the instructions of program using for users.

The advantages of this program are it provides a choice for user to choose either solving the VRPSD by ACS or SA algorithm and user may compare the efficiency between ACS and SA algorithm. By using the designed program, the operator may has a good planning for the pre-planned a priori tour in solid waste collection problem or delivery problem.

Finally, this chapter provides the introduction and applications of Programming with Microsoft Visual Studio C++ 6.0.

CHAPTER 7

THE EMULATION OF ANT COLONY SYSTEM AND SIMULATED ANNEALING FOR SINGLE VRPSD

Introduction

This chapter presents the development of ACS and SA for solving single VRPSD. In verifying the metaheuristics, a set of data is used

The Data

The customers' locations are in form of Cartesian coordinate where each point appears uniquely in a plane through two numbers, called x-coordinate and y-coordinate while demands of customers which are stochastic are recorded in a range form. Since there is no open source data for VRPSD problem, thus the data is adopted from a set of data modified from the well known 50-customer problem in Eilon *et al*. (1971) as in Table 7.1.

Table 7.1. Data from Elion et al. (1971)

i	**Customers' Locations**		ξ_i
	x-Coor.	**y-Coor.**	
0	5.00	4.00	–
1	1.89	0.77	4 – 7
2	9.27	1.49	2 – 9
3	9.46	9.36	3 – 5
4	9.20	8.69	0 – 4
5	7.43	1.61	5 – 7
6	6.08	1.34	0 – 8
7	5.57	4.60	3 – 5
8	6.10	2.77	3 – 6
9	8.99	2.45	1 – 7
10	8.93	7.00	6 – 7
11	8.60	0.53	5 – 7
12	4.01	0.31	1 – 6
13	3.34	4.01	3 – 7
14	6.75	5.57	4 – 7
15	7.36	4.03	1 – 2
16	1.24	6.69	1 – 3
17	7.13	1.92	1 – 7
18	7.86	8.74	4 – 6
19	4.18	3.74	5 – 9
20	2.22	4.35	5 – 6
21	0.88	7.02	5 – 8
22	8.53	7.04	1 – 5
23	6.49	6.22	1 – 8
24	4.53	7.87	1 – 3
25	4.46	7.91	0 – 4
26	2.83	9.88	3 – 5
27	3.39	5.65	1 – 6
28	0.75	4.98	0 – 5
29	7.55	5.79	1 – 4
30	8.45	0.69	1 – 6
31	3.33	5.78	3 – 7
32	6.27	3.66	3 – 8
33	7.31	1.61	0 – 6
34	6.37	7.02	1 – 6
35	7.23	7.05	0 – 7
36	1.68	6.45	1 – 4
37	3.54	7.06	2 – 8
38	7.67	4.17	0 – 6
39	2.2	1.12	3 – 7
40	3.57	1.99	2 – 8
41	7.34	1.38	0 – 4
42	6.58	4.49	1 – 6
43	5.00	9.00	1 – 4
44	6.63	5.23	3 – 4
45	5.89	8.06	1 – 8
46	1.13	5.25	2 – 6
47	1.90	8.35	3 – 7
48	1.74	1.37	2 – 9

The customers' locations are shown on a plane by designed program as in Figure 7.1 to show the positions of all customers graphically.

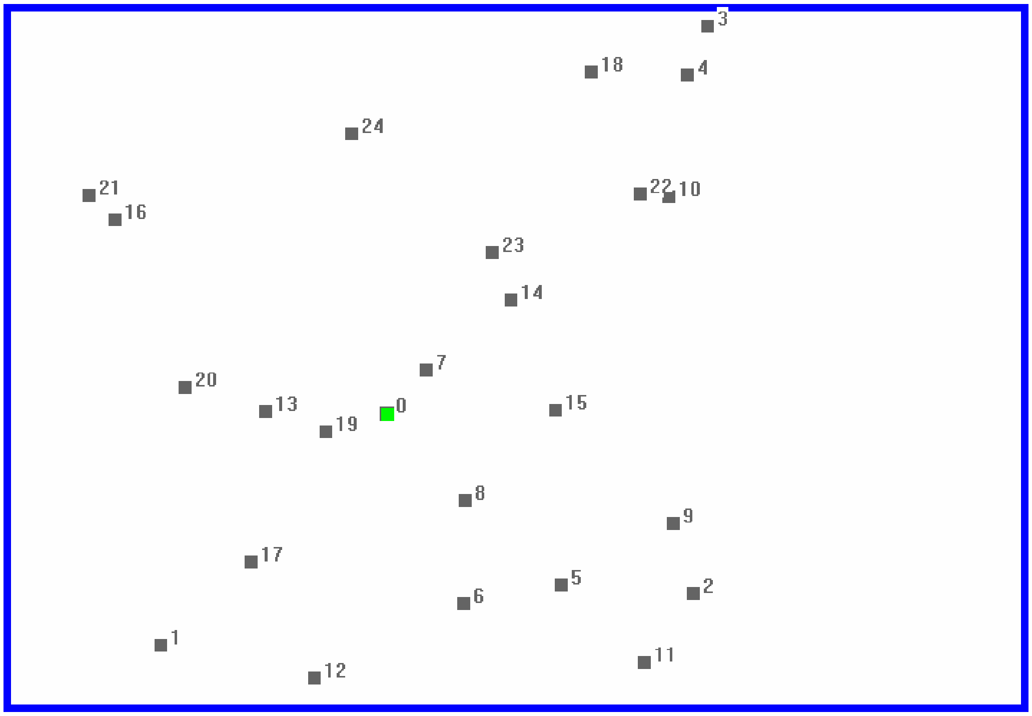


Figure 7.1. Positions of customers' locations

For this case study, the stopping criterion for ACS algorithm is in dynamic form where either number of consecutive non-improving solutions reaching 50 or number of iteration reaches the maximum level. For SA algorithm, the stopping criterion used is in static form where the iteration stops after a predetermined number of maximum iteration is reached. The move that is used in SA algorithm to obtain the neighboring solution is the swapping of position of any two randomly chosen customers' positions. Parameter settings of ACS and SA algorithm for this case study are summarized in Table 7.2 and Table 7.3 respectively.

Table 7.2. Parameter setting of ACS algorithm

Number of ants	10
Capacity of vehicle	10 – 50
a_0	0.9
α, ρ	0.1
β	2
τ_0	$(n.L_{nn})^{-1}$
Maximum iterations	100
Num. of consecutive non-improving solutions allowed	50

Table 7.3. Parameter setting of SA algorithm

Initial temperature	1000
Number of repetition	20
Cooling Rate	0.98
Maximum iteration	1500

7.3 Comparison of Algorithms over Problem Sizes

For this case study, the proposed ACS with local search algorithm and Simulated Annealing algorithm are tested for problems ranging in size from 12 to 48 customers. The program is run for 10 trials for each algorithm under different vehicle capacity values from ten to forty. Best cost refers to the best cost found from the 10 trials for comparison purpose. Table 5.4 summarizes the computational results for algorithm ACS and SA under different problem sizes.

Table 7.4. Comparison of algorithm ACS and SA for different problem sizes

Problem sizes, *N*	Capacity, *Q*	Total Expected Cost of a priori				Difference (Best ACS – Best SA)
		ACS		SA		
		Best	Average	Best	Average	
12 (Model 1)	10	69.4358	69.5276	69.4358	69.9263	0
	20	42.7758	42.7825	42.8547	43.2857	- 0.0789
	30	37.2618	37.3537	37.3588	37.6974	- 0.0970
	40	33.8966	33.9691	34.0416	34.7209	- 0.1450
24 (Model 2)	10	114.7422	114.8451	116.6947	117.9591	- 1.9525
	20	70.1857	70.5992	70.7853	72.9933	- 0.5996
	30	55.0705	55.5264	55.4437	58.4765	- 0.3732
	40	50.4478	50.9085	51.0495	53.1760	- 0.6017
36 (Model 3)	10	154.1018	154.8055	156.9437	159.3486	- 2.8419
	20	94.1029	94.9249	97.4506	99.4608	-3.3477
	30	73.5592	74.2729	77.1142	79.7676	- 3.5550
	40	62.6947	63.2958	68.7049	71.6983	- 6.0102
48 (Model 4)	10	199.2149	200.2451	209.8118	212.2541	- 10.5969
	20	118.7979	119.5347	126.0159	127.6119	- 7.2180
	30	91.1003	91.6274	96.3949	100.2883	- 5.2946
	40	76.9768	78.0332	84.0350	87.9365	- 7.0582

The results indicate that proposed ACS with local search algorithm produces better solutions compared to SA algorithm. For smaller size of problem, deviation between the best cost results from both ACS and SA algorithm is relatively small. For the problem size with 12 customers with vehicle capacity 10 units, both algorithms obtained the same best cost which is 69.4358 units. But the percentage deviations of averages from the associated best cost are 0.1322 and 0.7064 for ACS and SA. This indicates that ACS gives the results in a more consistent manner compared to SA.

The deviation for SA's solution compared to solutions obtained by ACS is increasing with the problem sizes. For problem size with 12 customers, the best solutions of SA deviated from best solutions of ACS for not more than 0.5 percent. When it comes to the problem sizes with 48 customers, the best solutions of SA deviated from best solutions of ACS for more than 5 percent. This indicates that

ACS performs much better than SA algorithm in bigger size of problem. However, ACS requires more computational effort or time over SA algorithm.

For each of the problem size, the larger of vehicle capacity gives the lower total expected cost. The reason behind is, the larger capacity of vehicle is able to satisfy more customers' demands in which it reduces the occurring times for preventive restocking and route failure.

In short, ACS showed better performance than SA under all of the problem sizes with various vehicle capacities tested. In next section, both algorithms will be tested over demand ranges. The complete set of result can be found in Appendix B.

7.4 Comparison of Algorithms over Demand Ranges

To access the robustness of ACS algorithm, both ACS and SA algorithms are further compared for different demand ranges. The customers' demands are generated ranged from 5 to 20. Again, the program is run for 10 trials for each algorithm under different demand ranges and problem sizes. The capacity of vehicle is set to be 30 units for every trial for this comparison. Table 7.5 summarizes the computational results for comparison.

Table 7.5. Comparison of algorithm ACS and SA for different demand ranges

Problem sizes, *N*	**Demand Ranges (max-min)**	**Total Expected Cost of a priori**				**Difference (Best ACS – Best SA)**
		ACS		**SA**		
		Best	**Average**	**Best**	**Average**	
12 (Model 1)	5	37.8309	37.8309	37.8309	37.8946	0
	10	46.5176	46.5233	46.5176	46.6858	0
	15	55.9193	55.9193	55.9193	56.4056	0
	20	66.3211	66.3906	66.4080	66.7586	- 0.0869

Table 7.5. Comparison of algorithm ACS and SA for different demand ranges (continued)

Problem sizes, ***N***	**Demand Ranges (max-min)**	**Total Expected Cost of a priori**				**Difference (Best ACS – Best SA)**
		ACS		**SA**		
		Best	**Average**	**Best**	**Average**	
24 (Model 2)	5	60.6567	60.8200	62.1815	63.3988	- 1.5248
	10	79.9053	80.0058	81.5747	80.6062	- 1.6694
	15	97.5548	97.6098	99.4316	101.3428	- 1.8768
	20	116.1023	116.1148	117.3202	119.0287	- 1.2179
36 (Model 3)	5	80.1849	80.8623	84.1031	85.5617	- 3.9182
	10	106.9875	107.3627	110.6752	112.5322	- 3.6877
	15	133.3562	133.5786	137.9571	141.2781	- 4.6009
	20	160.7067	161.0139	161.1994	166.7799	- 0.4927
48 (Model 4)	5	97.9829	98.7263	103.2490	105.2380	- 5.2661
	10	133.5311	134.0055	140.2354	143.1984	- 6.7043
	15	168.1742	168.8197	175.8930	179.7441	- 7.7188
	20	205.0472	205.5534	214.8841	218.3807	- 9.8369

The results indicate that for all demand ranges, proposed ACS algorithm showed better performance than SA algorithm. From table above, it can be noted that ACS gives the more consistent solutions compared to SA algorithm. The average total expected cost given by ACS does not deviate too far from its best cost.

With a fixed capacity, the deviation of best costs for SA algorithm from the best cost of ACS is increasing as the demand ranges increases. Further more, for problem size with 12 customers, the best solutions for SA deviated from best solutions of ACS for not more than 0.2 percent. However, for problem size with 48 customers, the best solutions of SA deviated from best solutions of ACS for more than 4.5 percent. This shows that ACS algorithm can reach a good solution for larger problem size compare to SA algorithm.

Apparently, SA is able to obtain good solutions for small ranges especially small size of problem. For ACS, it is always provide good results for all tested ranges and problems sizes of the tested problem. The complete set of result can be found in Appendix C.

7.5 Summary

The proposed ACS with local search algorithm and Simulated Annealing algorithm are tested for problems ranging in size from 12 to 48 customers. The results indicate that proposed ACS with local search algorithm produces better solutions compare to SA algorithm. Deviation of SA algorithm's best cost from the best cost results of ACS algorithm is increasing as the problem size increases.

In the second part of comparison, both ACS and SA algorithms are further compared for different demand ranges from 5 to 20 units. Again, the results indicate that proposed ACS algorithm showed better performance than SA algorithm. With a fixed capacity, the deviation of best costs for SA algorithm from the best cost of ACS is increasing as the demand ranges increases.

From the results of case study above, it can be known that the ACS algorithm always finds very good solutions for all tested problems in the aspects of various problem sizes and demand ranges. The algorithm finds the good solutions efficiently and consistently compare with other heuristic methods such as Simulated Annealing and it does not exhibit stagnation behavior where the ants continue to search for new possibly better tours. Stagnation behavior is the situation in which all ants make the same tour.

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 Introduction

This chapter provides a conclusion for this study based on the use of metaheuristics namely Tabu Search (TS), Genetic Algorithm (GA), Ant Colony System (ACS) and Simulated Annealing (SA) in solving Vehicle Routing Problem with Stochastic Demands (VRPSD). Some recommendations for future work and program improvements are included in this chapter.

8.2 Conclusions

The main purpose in this study is to study the applications of several metaheuristics in solving VRPSD. As this problem cannot be solved by optimal (exact) methods in practice, heuristics are used for this purpose.

The ACS exploits the nature phenomenon of ants to solve such stochastic optimization problem. A local search algorithm (Descent Method) is proposed to be added in the ACS algorithm. The concept of this method is to find the a priori tour that gives the minimum total expected cost. The ACS has been shown the ability in obtaining good solution for VRPSD problem.

Generally, Genetic Algorithm gives a pool of solutions rather than just one. The process of finding superior solutions mimics the evolution process, with solutions being combined or mutated to find out the pool of solutions. Simulated Annealing is a global optimization technique which traverses the search space by generating neighboring solutions of the current solution. A superior neighbor is always accepted and an inferior neighbor is accepted with some probability.

Tabu Search is similar to Simulated Annealing, in that both traverse the solution space by testing mutations of an individual solution. However, simulated annealing generates only one mutated solution but tabu search generates many mutated solutions and moves to the solution with the lowest fitness of those generated.

Ant Colony System is the extension from Ant System. Both algorithms are categorized as Ant Colony Optimization (ACO) algorithms. In particular, it can be observed that ACS is the most aggressive of the AS algorithms and it returns the best solution quality for very short computation times (Dorigo and Stutzle, 2004). ACS has an advantage over Simulated Annealing and Genetic Algorithm approaches when the graph may change dynamically where the Ant Colony algorithm can be run continuously and adapt to changes in real time.

8.3 Recommendations for Future Work

There are some aspects of this study can be improved such as metaheuristics approach, VRPSD objective function, and the program improvement. For Ant colony approach, it may be improved by letting more than one ant to contribute to the global updating rule to reduce the probability of being trapped in a local minimum and allowing the ants which produce very bad tours to subtract pheromone.

Besides, the current parallel local updating of pheromone may be changed to a sequential one. In ACS, all ants apply the local updating rule in parallel while they are building their tours. For the modified ACS, the first ant in the sequence starts builds its tour and it changes the pheromone on visited edges. After that, the second ant starts and so on until the last ant has built its tour. At this point, the global updating rule is applied. In this scheme, the preferred tour will tend to remain the same for all ants unlike in the ACS where local updating shuffles the tours. However, the search will be diversified since the later ant will search in a bigger neighborhood of the preferred tour (in fact, pheromone on the preferred tour decreases due to the evaporation as ants pass by, making it become less desirable for ants).

For further study, we may develop hybrid metaheuristics scheme that combine the strength of trajectory methods like Tabu Search and Simulated Annealing with population-based methods like Genetic Algorithm and Ant Colony Optimization in order to increase the effectiveness in getting optimal solution.

8.4 Suggestions for Program Improvement

1. The program may be improved by adding a “call form” that can guide users in step by step data key in.
2. The list controls which display the data information and result may be placed in another tab page for a wider display screen.
3. The compute button may add in the refinement ability so that the solution obtained may be improved by a second or few more clicks.

REFERENCES

Baker, B. M. and Ayechew, M. A. (2003). A Genetic Algorithm for the vehicle routing problem. *Computers & Operations Research.* 30: 787–800.

Barbarosoglu, G. and Ozgur, D. (1999). A Tabu Search algorithm for the vehicle routing problem. *Computers and Operations Research*. 26: 255 – 270.

Berger, J., Salois, M. and Begin, R. (1998). A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows. *Lecture Notes in Artificial Intelligence* 1418, AI'98, Advances in Artificial Intelligence, Vancouver, Canada, 114−127.

Bertsimas, D. J. (1992). A Vehicle Routing Problem with Stochastic Demand. *Operations Research.* 40 (3): 574-585.

Bertsimas, D. J. and Simchi-Levi, D. (1996). A New Generation of vehicle routing Research: Robust Algorithms, Addressing Uncertainty. *Operations Research*. 44 (2): 286-304.

Bertsimas, D., Jaillet, P. and Odoni, A. R. (1990). A Priori Optimization. *Operations Research.* 38 (6): 1019-1033.

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., and Schiavinotto, T. (2005). *Metaheuristics for the Vehicle Routing Problem with Stochastic Demands*. Technical Report No. IDSIA-06-05. IDSIA, Dalle Molle Institute for Artificial Intelligence. Switzerland.

Blanton, J.L. and Wainwright, R. L. (1993). Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, S. Forrest (editor), 452−459 Morgan Kaufmann, San Francisco.

Blum, C. and Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys.* 35 (3): 268-308

Bräysy, O., Berger, J. and Barkaoui, M. (2000). A New Hybrid Evolutionary Algorithm for the Vehicle Routing Problem with Time Windows. Presented in *Route 2000 Workshop*, Skodsborg, Denmark.

Bryant, K. (2000). *Genetic Algorithms and the Traveling Salesman Problem,* Department of Mathematics, Harvey Mudd College.

Chepuri, K. and Homem-De-Mello, T. (2005). Solving the Vehicle Routing Problem with Stochastic Demands using the Cross-Entropy Method. *Annals of Operations Research*. 134: 153-181.

Choi, In-Chan, Seng-In Kim and Hak-See Kim. (2003). A Genetic Algorithm with a mixed region search for the asymmetric TSP. *Computers and Operations Research.* 30: 773-786.

Coley, D. A. (1999). *An Introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publising Co. Pte. Ltd, Singapore.

Cordeau, J-F., Gendreau, M., Laporte, G., Potvin, J-Y., Semet, F., A Guide to Vehicle Routing Heuristics. *Journal of the Operational Research Society.* 53: 512-522.

Cordeau, J-F. and Laporte, G. (2002). *Tabu Search Heuristics for the Vehicle Routing Problem*. Canada Research Chair in Distribution Management and GERAD Report. Montreal. Canada.

Crainic, T. G., Gendreau, M., and Potvin, J-Y. (2005). Parallel Tabu Search. In Parallel Metaheuristics: A New Class of Algorithms. Edited by Enrique Alba. John Wiley and Sons, Inc. New Jersey.

Deneubourg, J.L., Goss, S., Aron, S. and Pasteels, J.M. (1989). Self-organized shortcuts in the Argentine Ant. *Naturwissenschaften* 76**.** 579-581.

Dorigo, M. and Caro, G. D. (1999). The Ant Colony Optimization Meta-Heuristic. In: Corne, D., Dorigo, M. and Glover, F. (Eds.) *New Ideas in Optimization* (pp.11-32). London: McGraw-Hill.

Dorigo, M.., Gambardella, L. M. (1997b). Ant colony system: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*. 1(1):53–66.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). The Ant System: Optimization by A Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*. 26(1):1-13.

Dorigo, M., and Stutzle, T. (2004). *Ant Colony Optimization*, Cambridge, Mass: MIT Press.

Dror, M., Laporte, G. and Trudeau, P. (1989). Vehicle Routing Problem with Stochastic Demands: Properties and Solution Frameworks. *Transportation Sciences*. 23 (3): 166-176.

Dror, M., Laporte, G. and Louveaux, F. V. (1993). Vehicle Routing with Stochastic Demands and Restricted Failures. *ZOR – Methods and Models of Operations Research*. 37: 273-283.

Gen, M. and Cheng, R. (1997). Genetic Algorithms and Engineering Design. John Wiley and Sons, Inc. Canada.

Gendreau, M. (2002). *An Introduction to Tabu Search*. Working paper. Departement d'informatique et de recherché operationnalle. Universite de Montreal. Montreal. Canada.

Gendreau, M., Hertz, A. and Laporte, G. (1994). A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*. 40: 1276 – 1290.

Gendreau, M., Laporte, G. and Seguin, R. (1996). A Tabu Search for the Vehicle Routing Problem with Stochastic Demands and Customers. *Operations Research*. 44 (3): 469-477.

Glover, F. (1990). Tabu Search: A Tutorial. *Interfaces*. 20: 74 – 94.

Glover, F. and M. Laguna. (1997). *Tabu Search*. Kluwer. Boston.

Glover, F. and Kochenberger, G. A. (2003). Handbook of Metaheuristics. Kluwer.

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

Jung, S. (2000). *A Genetic Algorithm for the Vehicle Routing Problem with Time-Dependent Travel Times.* PhD dissertation, University of Maryland.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 13 May 1983 220, 4598: 671-680.

Lacomme, P., Prins, C. and Sevaux, M. (2005). A Genetic Algorithm for a bi-objective capacitated arc routing problem. *Computers and Operations Research*.

Laporte, G. (1992). The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms. *European Journal of Operation Research.* 59: 345-358

Laporte, G., Louveaux, F. V. and Van Hamme, L. (2002). An Integer L-Shaped Algorithm for the Capacitated Vehicle Routing Problem with Stochastic Demands. *Operations Research*. 50 (3): 415-423.

Loft, B. and Snow, J. (2006). A Genetic Algorithm for Drawing Ordered Sets. *Texas College Mathematics Journal*. 1 (2): 10-25.

Metaheuristics Network Website. (2000). http://www.metaheuristics.net/.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge.

Obitko, M. (1998). An introduction to genetic algorithms with Java applets. http://cs.felk.cvut.cz/~xobitko/ga/.

Ortiz, F., Simpson, J. R., Pignatiello, J. J., and Langner, A. H. (2004). A Genetic Algorithm approach to Multiple-Response Optimization. *Journal of Quality Technology*, 36 (4).

Osman, I. H. (1993). Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problems. *Annals of Operations Research*. 41: 421-452

Osman, I. H. and Laporte, G. (1996). Metaheuristics: A Bibliography. *Annals of Operations Research*. 63: 513-623.

Potvin, J-Y. and Bengio, S. (1996). The Vehicle Routing Problem with Time Windows - Part II: Genetic Search. *INFORMS Journal on Computing.* 8:165–72.

Potvin J-Y, Dubre, D. and Robillard, C. (1996a). A hybrid approach to vehicle routing using neural networks and genetic algorithms. Applied Intelligence. 6 :241–52.

Potvin J-Y, Duhamel, C. and Guertin, F. (1996b). A Genetic Algorithm for vehicle routing with backhauling. *Applied Intelligence*. 6 :345–55.

Raidl, G. R. (2006). A Unified View on Hybrid Metaheuristics. In Francisco Almeida et al., editors, *Proceedings of the Hybrid Metaheuristics Workshop*, volume 4030 of *LNCS*, pages 1-12. Springer.

Randles, R. H. and Wolfe, D. A. (1979). Introduction to the Theory of Nonparamteric Statistics. John Wiley and Sons, Inc. USA.

Reeves, C. R. (1993). Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific Publishing. Oxford. England.

Rochat, Y. and Taillar, E. (1995). Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*. 1 : 147 – 167.

Salhi, S, Thangiah, S. R. and Rahman, F. (1998). A Genetic Clustering method for the multi-depot vehicle routing problem. In: Smith GD, Steel NC, Albrecht RF, editors. ICANNGA'97, Vienna. New York: Springer, 234–7.

Secomandi, N. (2001). A Rollout Policy for the Vehicle Routing Problem with Stochastic Demands. *Operations Research*. 49 (5): 796-802.

Silver, E. A., Vidal, R. V. and de Werra, D. (1980). A tutorial on heuristic methods. *European Journal of Operational Research*. 5 :153-162.

Stutze, T. (1999). Local Search Algorithms for Combinatorial Optimization: Analysis, Algorithms and New Applications. DISKI. Sankt Augustin, Germany.

Talbi, E.-G. (2002). A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*. 8: 541-564.

Tan, K.C., Lee, L.H. and Ou, K. (2001). Hybrid Genetic Algorithms in Solving Vehicle Routing Problems with Time Window Constraints. *Asia-Pacific Journal of Operational Research.* 18: 121−130.

Thangiah, S. R. (1995a). Vehicle Routing with Time Windows Using Genetic Algorithms. In *Application Handbook of Genetic Algorithms*: *New Frontiers*, *Volume II*, 253−277, L. Chambers (editor), CRC Press, Boca Raton.

Thangiah, S. R. (1995b). An Adaptive Clustering Method using a Geometric Shape for Vehicle Routing Problems with Time Windows. In *Proceedings of the 6th International Conference on Genetic Algorithms*, L.J. Eshelman (editor), 536−543 Morgan Kaufmann, San Francisco.

Thangiah, S. R. and Nygard, P. L. (1992). School bus routing using genetic algorithms. In: Proceedings of the SPIE Conference on Applications of Artificial Intelligence X: Knowledge Based Systems, Orlando, 387–97.

Tillman, F. A. (1969). The Multiple Terminal Delivery Problem with Probabilistic Demands. *Transportation Science*, 3: 192-204.

Yang, Wen-Huei. (1996). *Stochastic Vehicle Routing with Optimal Restocking*. PhD thesis. Case Western Reserve University.

Yang, Wen-Huei., Mathur, K. and Ballou, R. H. (2000). Stochastic Vehicle Routing Problem with Restocking. *Transportation Science*. 34 (1): 99-112

Appendix A
Computational results for numerical example

Iteration 1	City 0		City 1		City 2		City 3		City 4		City 5		City 0	Expected Cost
Ant 1	1	−	0	−	4	−	5	−	2	−	3	−	1	
			Phe[1][0]		Phe[0][4]		Phe[4][5]		Phe[5][2]		Phe[2][3]		Phe[3][1]	
			0.004274		0.004274		0.004274		0.004274		0.004274		0.004274	
Ant 2	4	−	3	−	0	−	1	−	2	−	5	−	4	
			Phe[4][3]		Phe[3][0]		Phe[0][1]		Phe[1][2]		Phe[2][5]		Phe[5][4]	
			0.004274		0.004274		0.004274		0.004274		0.004274		0.004274	
After Sorting														
Ant 1	0	−	4	−	5	−	2	−	3	−	1	−	0	59.8349
Ant 2	0	−	1	−	2	−	5	−	4	−	3	−	0	56.5243
Descent Method														
Ant 1	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Ant 2	0	−	2	−	1	−	4	−	5	−	3	−	0	55.0749
Global Best	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Route			Phe[0][4]		Phe[4][5]		Phe[5][3]		Phe[3][2]		Phe[2][1]		Phe[1][0]	
			0.005677		0.005677		0.005677		0.005677		0.005677		0.005677	

Iteration 2	City 0		City 1		City 2		City 3		City 4		City 5		City 0	Expected Cost
Ant 1	4	−	0	−	1	−	2	−	3	−	5	−	4	
			Phe[4][0]		Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][4]	
			0.005537		0.005410		0.005537		0.005537		0.005537		0.005410	
Ant 2	3	−	1	−	0	−	4	−	5	−	2	−	3	
			Phe[3][1]		Phe[1][0]		Phe[0][4]		Phe[4][5]		Phe[5][2]		Phe[2][3]	
			0.004274		0.005410		0.005410		0.005537		0.004274		0.005410	
After Sorting														
Ant 1	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4578
Ant 2	0	−	4	−	5	−	2	−	3	−	1	−	0	59.8349
Descent Method														
Ant 1	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Ant 2	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Global Best	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Route			Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][4]		Phe[4][0]	
			0.006706		0.006819		0.006706		0.006819		0.006706		0.006706	

Iteration 3	City 0		City 1		City 2		City 3		City 4		City 5		City 0	Expected Cost
Ant 1	3	−	0	−	1	−	5	−	4	−	2	−	3	
			Phe[3][0]		Phe[0][1]		Phe[1][5]		Phe[5][4]		Phe[4][2]		Phe[2][3]	
			0.004274		0.006243		0.004274		0.006462		0.004274		0.006462	
Ant 2	4	−	0	−	1	−	2	−	5	−	3	−	4	
			Phe[4][0]		Phe[0][1]		Phe[1][2]		Phe[2][5]		Phe[5][3]		Phe[3][4]	
			0.006462		0.006244		0.006565		0.004274		0.006565		0.004274	
After Sorting														
Ant 1	0	−	1	−	5	−	4	−	2	−	3	−	0	59.6943
Ant 2	0	−	1	−	2	−	5	−	3	−	4	−	0	60.8753
Descent Method														
Ant 1	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Ant 2	0	−	2	−	1	−	4	−	5	−	3	−	0	55.0749
Global Best	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Route			Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][4]		Phe[4][0]	
			0.007456		0.007745		0.007652		0.007745		0.007652		0.007652	

Iteration 4	City 0		City 1		City 2		City 3		City 4		City 5		City 0	Expected Cost
Ant 1	2	−	0	−	4	−	5	−	3	−	1	−	2	
			Phe[2][0]		Phe[0][4]		Phe[4][5]		Phe[5][3]		Phe[3][1]		Phe[1][2]	
			0.004274		0.007011		0.007315		0.007397		0.004274		0.007085	
Ant 2	0	−	4	−	1	−	2	−	3	−	5	−	0	
			Phe[0][4]		Phe[4][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][0]	
			0.007315		0.004274		0.007397		0.007314		0.007085		0.004274	
After Sorting														
Ant 1	0	−	4	−	5	−	3	−	1	−	2	−	0	55.2670
Ant 2	0	−	4	−	1	−	2	−	3	−	5	−	0	61.4526
Descent Method														
Ant 1	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Ant 2	0	−	2	−	1	−	4	−	5	−	3	−	0	55.0749
Global Best	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Route			Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][4]		Phe[4][0]	
			0.008546		0.008213		0.008419		0.008213		0.008419		0.008146	

Iteration 5	City 0		City 1		City 2		City 3		City 4		City 5		City 0	Expected Cost
Ant 1	0	−	4	−	2	−	3	−	5	−	1	−	0	
			Phe[0][4]		Phe[4][2]		Phe[2][3]		Phe[3][5]		Phe[5][1]		Phe[1][0]	
			0.007759		0.004274		0.008005		0.007819		0.004274		0.007734	
Ant 2	5	−	4	−	0	−	1	−	2	−	3	−	5	
			Phe[5][4]		Phe[4][0]		Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]	
			0.008005		0.007410		0.008119		0.007819		0.007632		0.007464	
After Sorting														
Ant 1	0	−	4	−	2	−	3	−	5	−	1	−	0	67.4628
Ant 2	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Descent Method														
Ant 1	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Ant 2	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Global Best	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Route			Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][4]		Phe[4][0]	
			0.008797		0.008873		0.008705		0.008554		0.009041		0.008505	

Iteration 6	City 0		City 1		City 2		City 3		City 4		City 5		City 0	Expected Cost
Ant 1	4	−	0	−	1	−	2	−	5	−	3	−	4	
			Phe[4][0]		Phe[0][1]		Phe[1][2]		Phe[2][5]		Phe[5][3]		Phe[3][4]	
			0.008082		0.008345		0.008413		0.004274		0.008126		0.004274	
Ant 2	5	−	4	−	0	−	1	−	2	−	3	−	5	
			Phe[5][4]		Phe[4][0]		Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]	
			0.008564		0.007701		0.007938		0.007999		0.008262		0.007741	
After Sorting														
Ant 1	0	−	1	−	2	−	5	−	3	−	4	−	0	60.8753
Ant 2	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Descent Method														
Ant 1	0	−	2	−	1	−	4	−	5	−	3	−	0	55.0749
Ant 2	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Global Best	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Route			Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][4]		Phe[4][0]	
			0.008980		0.009036		0.009272		0.008803		0.009544		0.008768	

Iteration 7	City 0		City 1		City 2		City 3		City 4		City 5		City 0	Expected Cost
Ant 1	3	−	0	−	1	−	2	−	4	−	5	−	3	
			Phe[3][0]		Phe[0][1]		Phe[1][2]		Phe[2][4]		Phe[4][5]		Phe[5][3]	
			0.004274		0.008086		0.008560		0.004274		0.008543		0.007943	
Ant 2	1	−	0	−	4	−	5	−	3	−	2	−	1	
			Phe[1][0]		Phe[0][4]		Phe[4][5]		Phe[5][3]		Phe[3][2]		Phe[2][1]	
			0.008510		0.008318		0.009017		0.008350		0.008772		0.008131	
After Sorting														
Ant 1	0	−	1	−	2	−	4	−	5	−	3	−	0	55.2604
Ant 2	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Descent Method														
Ant 1	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Ant 2	0	−	4	−	5	−	3	−	2	−	1	−	0	54.6327
Global Best	0	−	1	−	2	−	3	−	5	−	4	−	0	54.4587
Route			Phe[0][1]		Phe[1][2]		Phe[2][3]		Phe[3][5]		Phe[5][4]		Phe[4][0]	
			0.009114		0.009154		0.009731		0.008985		0.009525		0.009323	

FINAL SOLUTION

Global Best Route (a priori):

0 − 1 − 2 − 3 − 5 − 4 − 0

Total Expected Cost = 54.4587

Global Best Route with Expected Demands

0 − 1 − 2 − 3 − 0 − 5 − 4 − 0

Total Expected Cost = 52.936

Appendix B
Computational result for different problem sizes

	Capacity (10)	ACS	SA	Capacity (20)	ACS	SA
n = 12	1	69.4358	69.8700	1	42.7758	42.8589
	2	69.6467	69.9069	2	42.7850	43.8863
	3	69.5435	69.4385	3	42.7850	43.8628
	4	69.4575	70.4907	4	42.7758	42.8874
	5	69.5435	69.9069	5	42.7850	43.8830
	6	69.4575	69.5435	6	42.7850	42.8589
	7	69.6467	69.5435	7	42.7785	42.8547
	8	69.5435	70.0882	8	42.7850	42.8547
	9	69.5435	70.3869	9	42.7850	42.9326
	10	69.4575	70.0882	10	42.7850	43.9773
	Average	69.5276	69.9263	Average	42.7825	43.2857
	Best	69.4358	69.4385	Best	42.7758	42.8547
	Capacity (30)	**ACS**	**SA**	**Capacity (40)**	**ACS**	**SA**
	1	37.2618	37.8461	1	33.8966	34.0416
	2	37.2618	37.6607	2	34.0416	34.4675
	3	37.6209	37.8466	3	34.0416	34.4337
	4	37.5149	37.6761	4	33.8966	34.6327
	5	37.2618	37.7998	5	34.0416	34.6327
	6	37.5683	37.6607	6	33.8966	35.2914
	7	37.2618	37.3588	7	33.8966	35.0907
	8	37.2618	37.8461	8	34.0416	35.4416
	9	37.2618	37.7643	9	34.0416	34.6327
	10	37.2618	37.5149	10	33.8966	34.5439
	Average	37.3537	37.6974	Average	33.9691	34.7209
	Best	37.2618	37.3588	Best	33.8966	34.0416

	Capacity (10)	ACS	SA	Capacity (20)	ACS	SA
n = 24	1	115.6314	116.7344	1	70.6635	72.0196
	2	114.7622	117.5800	2	70.1857	72.1466
	3	114.7622	117.8070	3	70.6635	73.8640
	4	114.7622	118.2235	4	70.6635	74.8441
	5	114.7422	116.9303	5	70.6524	73.0754
	6	114.7622	116.6947	6	70.5987	72.5846
	7	114.7622	117.7313	7	70.5833	70.7853
	8	114.7622	119.6937	8	70.6635	73.4832
	9	114.7622	119.5678	9	70.7217	74.8438
	10	114.7422	118.6287	10	70.5959	72.2866
	Average	114.8451	117.9591	Average	70.5992	72.9933
	Best	114.7422	116.6947	Best	70.1857	70.7853
	Capacity (30)	**ACS**	**SA**	**Capacity (40)**	**ACS**	**SA**
	1	55.8752	56.5617	1	50.8439	52.3562
	2	55.3715	59.7675	2	50.8439	51.4846
	3	55.0907	61.7211	3	51.3761	54.3662
	4	55.2055	59.0000	4	51.3761	53.7147
	5	55.0705	59.5702	5	50.8439	55.1644
	6	55.7971	57.4225	6	50.4478	53.0544
	7	56.2181	57.2364	7	50.7826	51.0495
	8	55.0907	60.0380	8	50.8439	52.9504
	9	55.8752	57.0041	9	50.9149	55.6137
	10	55.6693	56.4437	10	50.8122	52.0059
	Average	55.5264	58.4765	Average	50.9085	53.1760
	Best	55.0705	56.4437	Best	50.4478	51.0495

	Capacity (10)	**ACS**	**SA**	**Capacity (20)**	**ACS**	**SA**
n = 36	1	154.6367	160.1961	1	94.6839	99.8777
	2	154.9405	158.6371	2	94.9025	100.9168
	3	154.2722	158.7943	3	94.9767	97.4506
	4	155.5823	159.4611	4	94.7980	98.5450
	5	154.1018	159.3102	5	95.4961	100.0834
	6	154.8023	159.2105	6	95.5011	98.3040
	7	155.0519	159.5609	7	94.1231	99.9228
	8	154.4073	162.9857	8	95.1888	98.4287
	9	155.0003	158.3868	9	94.1029	99.5134
	10	155.2596	156.9437	10	95.4760	101.5651
	Average	154.8055	159.3486	Average	94.9249	99.4608
	Best	154.1018	156.9437	Best	94.1029	97.4506
	Capacity (30)	**ACS**	**SA**	**Capacity (40)**	**ACS**	**SA**
	1	74.3692	83.7802	1	63.2139	68.7835
	2	74.5966	77.3196	2	63.2328	70.5108
	3	73.9676	77.6087	3	63.0388	68.7049
	4	74.1547	79.6420	4	63.1630	75.7079
	5	73.5592	78.6749	5	64.1176	72.9964
	6	74.1825	80.6051	6	62.6947	73.4751
	7	75.0294	80.6868	7	63.6378	73.8138
	8	73.8329	81.1141	8	63.2508	71.9439
	9	74.4678	77.1142	9	63.5032	69.2949
	10	74.5686	81.1299	10	63.1055	71.7519
	Average	74.2729	79.7676	Average	63.2958	71.6983
	Best	73.5592	77.1142	Best	62.6947	68.7049

	Capacity (10)	**ACS**	**SA**	**Capacity (20)**	**ACS**	**SA**
n = 48	1	200.7342	216.2808	1	118.7979	129.4671
	2	200.0541	213.1050	2	119.8071	126.2883
	3	199.2149	213.6705	3	119.5007	126.0159
	4	200.3239	211.9459	4	119.3693	128.2230
	5	200.4473	213.4700	5	119.4437	126.4835
	6	200.1959	211.7568	6	119.7060	127.6128
	7	199.8048	209.8118	7	119.2933	129.2605
	8	200.4827	211.1829	8	119.5128	127.0783
	9	200.0273	211.0323	9	119.5641	126.7557
	10	201.1663	210.2848	10	120.3522	128.9342
	Average	200.2451	212.2541	Average	119.5347	127.6119
	Best	199.2149	209.8118	Best	118.7979	126.0159
	Capacity (30)	**ACS**	**SA**	**Capacity (40)**	**ACS**	**SA**
	1	91.2987	100.7316	1	78.1829	84.0867
	2	91.5189	97.0785	2	79.3764	86.3547
	3	91.7264	100.4024	3	79.0092	92.8496
	4	92.2051	101.1812	4	76.9768	89.0924
	5	91.3195	100.8146	5	77.4840	85.0039
	6	91.1003	101.4224	6	77.6929	88.2642
	7	91.3816	96.3949	7	77.2585	88.7771
	8	91.7970	98.9237	8	77.6946	84.0350
	9	91.7830	101.0227	9	78.1219	89.4476
	10	92.1431	104.9112	10	78.5346	91.4540
	Average	91.6274	100.2883	Average	78.0332	87.9365
	Best	91.1003	96.3949	Best	76.9768	84.0350

Appendix C
Computational result for different demand ranges

	Dmd Range (5)	ACS	SA	Dmd Range (10)	ACS	SA
n = 12	1	37.8309	37.8309	1	46.5206	46.8988
	2	37.8309	37.8488	2	46.5206	46.5206
	3	37.8309	37.8630	3	46.5176	46.6386
	4	37.8309	37.9658	4	46.5176	46.6898
	5	37.8309	37.8345	5	46.5206	46.6386
	6	37.8309	37.9292	6	46.5206	46.5176
	7	37.8309	37.9591	7	46.5206	46.6577
	8	37.8309	37.9457	8	46.5386	46.5176
	9	37.8309	37.9271	9	46.5176	46.9898
	10	37.8309	37.8414	10	46.5386	46.7891
	Average	37.8309	37.8946	Average	46.5233	46.6858
	Best	37.8309	37.8309	Best	46.5176	46.5176
	Dmd Range (15)	ACS	SA	Dmd Range (20)	ACS	SA
	1	55.9193	56.0396	1	66.4080	67.0337
	2	55.9193	56.8359	2	66.3211	66.5212
	3	55.9193	56.3900	3	66.4080	66.4080
	4	55.9193	56.3800	4	66.4080	66.4080
	5	55.9193	56.2987	5	66.4080	66.4641
	6	55.9193	56.1189	6	66.3211	67.4146
	7	55.9193	55.9193	7	66.4080	66.7276
	8	55.9193	57.1906	8	66.4080	67.6057
	9	55.9193	55.9193	9	66.4080	66.5949
	10	55.9193	56.9635	10	66.4080	66.4080
	Average	55.9193	56.4056	Average	66.3906	66.7586
	Best	55.9193	55.9193	Best	66.3211	66.4080

	Dmd Range (5)	ACS	SA	Dmd Range (10)	ACS	SA
n = 24	1	60.6733	63.2559	1	79.9689	81.7906
	2	60.6567	64.4601	2	80.1622	81.5747
	3	60.6567	63.9817	3	80.0595	82.2676
	4	61.1208	62.5469	4	79.9053	82.9661
	5	60.6567	62.9075	5	79.9589	82.2295
	6	60.6567	62.1815	6	79.9589	83.5011
	7	60.6567	63.3900	7	80.0149	83.9647
	8	60.9831	64.1865	8	79.9053	82.0714
	9	61.4777	63.8513	9	80.1086	83.0139
	10	60.6611	63.2263	10	80.0159	82.6824
	Average	60.8200	63.3988	Average	80.0058	82.6062
	Best	60.6567	62.1815	Best	79.9053	81.5747
	Dmd Range (15)	ACS	SA	Dmd Range (20)	ACS	SA
	1	97.5562	100.7251	1	116.1232	120.2019
	2	97.6145	101.9586	2	116.1023	119.7339
	3	97.6145	100.3302	3	116.1023	118.8899
	4	97.6145	101.9371	4	116.1023	118.1318
	5	97.6145	99.4316	5	116.1023	118.0793
	6	97.6145	102.1688	6	116.1438	119.9205
	7	97.5548	100.7495	7	116.1023	119.5699
	8	97.6858	102.1522	8	116.1023	119.5374
	9	97.6145	101.0403	9	116.1438	118.9024
	10	97.6145	102.9346	10	116.1232	117.3202
	Average	97.6098	101.3428	Average	116.1148	119.0287
	Best	97.5548	99.4316	Best	116.1023	117.3202

n = 12	**Dmd Range (5)**	**ACS**	**SA**	**Dmd Range (10)**	**ACS**	**SA**
	1	37.8309	37.8309	1	46.5206	46.8988
	2	37.8309	37.8488	2	46.5206	46.5206
	3	37.8309	37.8630	3	46.5176	46.6386
	4	37.8309	37.9658	4	46.5176	46.6898
	5	37.8309	37.8345	5	46.5206	46.6386
	6	37.8309	37.9292	6	46.5206	46.5176
	7	37.8309	37.9591	7	46.5206	46.6577
	8	37.8309	37.9457	8	46.5386	46.5176
	9	37.8309	37.9271	9	46.5176	46.9898
	10	37.8309	37.8414	10	46.5386	46.7891
	Average	37.8309	37.8946	Average	46.5233	46.6858
	Best	37.8309	37.8309	Best	46.5176	46.5176
	Dmd Range (15)	**ACS**	**SA**	**Dmd Range (20)**	**ACS**	**SA**
	1	55.9193	56.0396	1	66.4080	67.0337
	2	55.9193	56.8359	2	66.3211	66.5212
	3	55.9193	56.3900	3	66.4080	66.4080
	4	55.9193	56.3800	4	66.4080	66.4080
	5	55.9193	56.2987	5	66.4080	66.4641
	6	55.9193	56.1189	6	66.3211	67.4146
	7	55.9193	55.9193	7	66.4080	66.7276
	8	55.9193	57.1906	8	66.4080	67.6057
	9	55.9193	55.9193	9	66.4080	66.5949
	10	55.9193	56.9635	10	66.4080	66.4080
	Average	55.9193	56.4056	Average	66.3906	66.7586
	Best	55.9193	55.9193	Best	66.3211	66.4080

n = 24	**Dmd Range (5)**	**ACS**	**SA**	**Dmd Range (10)**	**ACS**	**SA**
	1	60.6733	63.2559	1	79.9689	81.7906
	2	60.6567	64.4601	2	80.1622	81.5747
	3	60.6567	63.9817	3	80.0595	82.2676
	4	61.1208	62.5469	4	79.9053	82.9661
	5	60.6567	62.9075	5	79.9589	82.2295
	6	60.6567	62.1815	6	79.9589	83.5011
	7	60.6567	63.3900	7	80.0149	83.9647
	8	60.9831	64.1865	8	79.9053	82.0714
	9	61.4777	63.8513	9	80.1086	83.0139
	10	60.6611	63.2263	10	80.0159	82.6824
	Average	60.8200	63.3988	Average	80.0058	82.6062
	Best	60.6567	62.1815	Best	79.9053	81.5747
	Dmd Range (15)	**ACS**	**SA**	**Dmd Range (20)**	**ACS**	**SA**
	1	97.5562	100.7251	1	116.1232	120.2019
	2	97.6145	101.9586	2	116.1023	119.7339
	3	97.6145	100.3302	3	116.1023	118.8899
	4	97.6145	101.9371	4	116.1023	118.1318
	5	97.6145	99.4316	5	116.1023	118.0793
	6	97.6145	102.1688	6	116.1438	119.9205
	7	97.5548	100.7495	7	116.1023	119.5699
	8	97.6858	102.1522	8	116.1023	119.5374
	9	97.6145	101.0403	9	116.1438	118.9024
	10	97.6145	102.9346	10	116.1232	117.3202
	Average	97.6098	101.3428	Average	116.1148	119.0287
	Best	97.5548	99.4316	Best	116.1023	117.3202