DESIGN AND ANALYSIS OF ALGORITHMS FOR SOLVING SOME STOCHASTIC VEHICLE ROUTING AND SCHEDULING PROBLEMS

TENG SUYAN

NATIONAL UNIVERSITY OF SINGAPORE

2004

DESIGN AND ANALYSIS OF ALGORITHMS FOR SOLVING SOME STOCHASTIC VEHICLE ROUTING AND SCHEDULING PROBLEMS

By

TENG SUYAN (B.ENG. M.ENG.)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2004

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my supervisors, Associate Professor Ong Hoon Liong and Associate Professor Huang Huei Chuen, who provided patient guidance and constant encouragement throughout the study and research process. I would also like to thank all other faculty members of the ISE Department, from whom I have learnt a lot through coursework and seminars.

Special gratitude also goes to those colleagues who accompanied me and made my stay in the Department pleasant and memorable. Particularly, I am grateful to Lin shenxue, Gao yinfeng, Yang guiyu, Liu shubin, Yew Loon, Adam, Mong Soon, Liang zhe, Ivy, who kindly offered help in one way or another. Also I would like to extend my thanks to those whose names are not listed here, for their concern and help.

A special thank is for my mother who always cared and loved me with all her heart.

This dissertation is dedicated to my husband, Mr. Wang zhidong, and my daughter, Wang qing. They gave me all the love and encouragement when I was in the low moments that inevitably occurred during the development of the dissertation. Lastly, but not the least, I would like to thank my father, my parents-in-law and all members of my family for their continuous encouragement and support.

TENG SUYAN

TABLE OF CONTENTS

SUMMARY

The classical traveling salesman problem (TSP) is the most studied combinatorial NP-hard problem in the literature. This thesis addresses two variants of the TSP: the vehicle routing problem with stochastic demands (VRPSD) and the time constrained traveling salesman problem (TCTSP).

For the VRPSD, the problem is studied based on the formulation of stochastic programming with recourse, which is within the framework of *a priori* optimization. A comparative study among heuristics available in the literature is firstly carried out to determine which one is superior to the others in a certain context; and valuable suggestions and recommendations are made for decision makers in various scenarios. Secondly, as most of the heuristics presented in the literature belong to classical local search algorithms, the thesis proposes three metaheuristics: simulated annealing (SA), threshold accepting (TA) and tabu search (TS), to examine whether metaheuristics are more preferable for the VRPSD, and which metaheuristic is superior to the others in a certain context. Computational results show that, metaheuristics can obtain solutions with better solution quality for VRPSD, though they may consume more computational time.

For the TCTSP, we first extend it into a multi-period problem: find a subset of customers as well as the sequence of serving them in each period in a rolling schedule environment, so that the average profit per period in the long run is maximized.

Several heuristics based on iterative customer assignment and iterative centre-of-gravity have been proposed for solving the problem. Then, the problem is formulated as a set-covering problem and its linear programming relaxation is solved to optimality by a column generation scheme to get an upper bound. To evaluate the performance of the heuristics, for small size problems with long service time, the heuristics are compared against the upper bound; for other cases, they are compared among themselves. Computational results illustrate that, the best representative of each heuristic performs very well for the problem, with the largest average percentage deviation from the upper bound being 2.24%, and the smallest deviation only 1.02%. When comparing the heuristics among themselves, results indicate that, with respect to solution quality, each heuristic has its own advantage in a certain scenario. Decision makers are advised to employ different heuristics in different scenarios. Secondly, the TCTSP is further extended into the stochastic case, where the travel and service times are assumed to be independent random variables. This extension is important because: (a) Both travel and service times are not likely to be deterministic in the practical situations; (b) The profit generated from visiting a subset of the customers is directly affected by the travel and service times due to the time limit constraint. Again, within the framework of *a priori* optimization, two models are proposed for formulating the problem: a chance-constrained program and a stochastic program with recourse. Then an integer L-shaped solution method is developed to solve the problem to optimality. Results show that, the proposed algorithm can solve the stochastic TCTSP with moderate problem size to optimality within reasonable time.

NOMENCLATURE

SVRP	Stochastic vehicle routing problem
VRPSD	Vehicle routing problem with stochastic demands
SPR	Stochastic program with recourse
TCTSP	Time constrained traveling salesman problem
LP	Linear programming
n	Problem size - the number of customers
V	$\{1, \ldots, n\}$ denotes a set of n customers
V'	$V \cup \{0\}$ denotes a set of nodes including customers and the depot
A	$\{(i, j) \mid i, j \in V'$ and $i < j\}$ denotes a set of arcs
$c_{i,j}$	Traveling distance between customer i and j.
m	Number of vehicles
Q	Vehicle capacity
D_i	A random variable that describes the demand of customer i
L	A predefined maximum limit for the expected route length.
γ_i	The probability that the demand at the i^{th} node is exactly equal to the stock available in the vehicle
δ_i	The probability that the demand at the i^{th} node exceeds the remaining stock
q	The vehicle's remaining load

$p_i(k)$	The probability that demand at node i is k.
$q^*(j)$	The threshold value for node j. If the remaining load after visiting node j is less than $q^*(j)$, it is better to return to the depot before serving further demand points.
t_0	Initial temperature of a SA algorithm
t_f	Final temperature of a SA algorithm
α	Temperature reduction factor
ε	Thermal equilibrium factor
$\bar{m}$	An upper bound for the number of vehicles m required
ϖ	Neighbor list size
π	An ordered sequence starting and ending at depot
$L(\pi)$	The expected route length of route π
ρ	A positive parameter to penalize the objective function due to the fact that the expected route length exceeds the predefined L
g_j	The date a customer j requires the service
σ_j	Tolerance time of a customer j
R_j	Profit of visiting customer j
H	Number of periods in a rolling horizon
$[e_j, l_j]$	Time window that customer j can be visited
p_{ij}	Profit of customer j if it is assigned to period i
w_{ij}	Weight of customer j if it is assigned to period i
T	The effective working time for each period

T_i	The remaining working time of period *i*
f_{ij}	A measure of the desirability of assigning customer *j* into period *i*
λ	Mean arrival rate of a Poisson distribution
$G_i(x, y)$	Center-of-gravity of the tour in period *i*
Ω_i	The set of all possible sub-tours in period *i*
t_{ij}	A random variable representing time of traveling arc (*i*, *j*)
τ_j	A random variable representing service time of visiting node *j*
α	Maximum probability that total travel and service time of a tour is allowed to exceed *T*
ΔT	Maximum amount of time allowed to exceed *T*
ξ	A vector of random variables corresponding to travel and service time
$\theta(\xi^k)$	Total travel and service time of the route in excess of *T* when the realization of the random variable is ξ^k
$t_{ij}^{\xi^k}$	A random variable representing time of traveling arc (*i*, *j*) when the realization of the random variable is ξ^k
$\tau_j^{\xi^k}$	A random variable representing service time of visiting node *j* when the realization of the random variable is ξ^k
β	The unit penalty cost for total time of the route in excess of *T*
η	A bound to estimate the expected penalty incurred for a given solution

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Given a set of cities, the classical traveling salesman problem (TSP) tries to determine a minimal cost cycle that passes through each node exactly once and starts and ends at the same city. In this dissertation, two variants of the TSP are considered: the vehicle routing problem with stochastic demands (VRPSD), and the time constrained traveling salesman problem (TCTSP).

In the first problem, a fleet of vehicles with limited capacity are assumed to deliver goods to the customers from the depot. The demands of the customers are defined as random variables, because they are not known when constructing the vehicle routes. The problem is to determine vehicle routes so that total expected distance traveled by the vehicles is minimized while satisfying some side constraints. In the second problem, it is assumed that each customer is associated with a profit of visiting it. Given a predefined effective working time limit, the problem tries to maximize the profit generated from visiting the customers while satisfying the time limit constraint. The TCTSP is a relaxed variant of the TSP or a generalized TSP (GTSP) in the sense that not all customers are needed to be visited due to the time limit constraint imposed on the time duration of the tour. For the TCTSP, this study first considers a multi-period TCTSP in a rolling schedule environment; then it is extended into the stochastic case: a TCTSP with stochastic travel and service times.

1. 1 Introduction to the Stochastic Vehicle Routing Problems

The management of a distribution system involves many problems, such as administration problems in running the depots, in designing an information system, in routing and scheduling of vehicles to customers, in loading of goods into vehicles and so on. The vehicle routing problem (VRP), which requires routing and scheduling the vehicles to perform the assigned functions at minimal cost, lies at the center of the management of a distribution system. Typically, the problem involves bringing products located at a central facility (where vehicles of limited capacity are also assumed to be initially housed) to geographically dispersed facilities at minimum cost, while satisfying various side constraints. This area of study, which mainly consists of designing optimum-seeking algorithms to identify the best configuration of routes and schedules, has become a very hot research topic and has been extensively studied by many operations researchers. Excellent surveys in this area can be found in Lawler *et al.* (1985) on the traveling salesman problem, Bodin *et al.* (1983) for routing and scheduling, and Golden and Assad (1988), Laporte (1992) and Fisher (1996) on vehicle routing problems.

The capacitated vehicle routing problem (VRP) plays an important role in distribution management and has been both extensively studied by researchers and applied in practice. The VRP can be broadly classified into two classes of problems: the deterministic VRP and the stochastic VRP (SVRP). For the deterministic VRP, all the problem parameters, such as demands, travel cost and customer presence, are assumed to be known with certainty. For the stochastic VRP, in contrast, one or some components of the problem parameters may not be known for sure. The problem of constructing vehicle routes through the customers that minimizes the expected distance traveled is known as the SVRP. The SVRP has received increasing attention in recent years. Depending on which element is stochastic, the SVRP can be further divided into the following categories.

1) The probabilistic traveling salesman problem (PTSP)

   Introduced by Jaillet (1985, 1988), the PTSP is also known as the traveling salesman problem with stochastic customers (TSPSC), where each vertex v_i is present with probability p_i.

2) The traveling salesman problem with stochastic traveling times (TSPST)

   In the TSPST, the traveling time between any two customers is a random variable. In the case when *m*-vehicles are scheduled to visit a set of customers, the problem becomes *m*-TSPST.

3) The vehicle routing problem with stochastic customers (VRPSC)

   In the VRPSC, customers are present with some probability but have deterministic demands. It is an extension of the PTSP, where the vehicle capacity constraint must be satisfied; and once the vehicle capacity is attained or exceeded, the vehicle may have to go back to depot. This problem is well studied in Bertsimas (1988).

4) The vehicle routing problem with stochastic demands (VRPSD)

   In the VRPSD, customer demands are not known with certainty in advance; they are usually assumed to be independent random variables with known probability distributions. The VRPSD is the most studied problem in SVRP in the literature.

5) The vehicle routing problem with stochastic customers and demands (VRPSCD)

   The VRPSCD is a combination of both VRPSD and VRPSC, which means that, not only the customers are present with a certain probability, their demands are also random variables. It is an extremely difficult problem; even computing the value of the objective function is hard (Bertsimas 1992, Gendreau *et al.* 1996b).

As the most studied problem in SVRP, VRPSD has been employed to model and provide solutions for many real-world problems in practice. In Bertsimas (1992), the application areas identified include the distribution of packages from a post office, routing of forklifts in a cargo terminal or in a warehouse, and strategic planning of a delivery and collection company which has decided to begin service in a particular area. In Yang *et al.* (2000), the applications cover the following areas: constructing waste collection routes with volume of trash at each stop being unknown; delivery of money to automatic teller machines from a central bank; peddle routes construction, such as beer distribution to retail outlets, resupply of baked goods at food stores, replenishment of liquid gas at research laboratories, and stocking of vending machines, etc. Other cited applications in the literature include: delivery of money to branches or automatic teller machines of a central bank (Lambert *et al.*, 1993), less than truckload operations (Gendreau *et al.*, 1995), the delivery of home heating oil (Dror *et al.*, 1985), sludge disposal, where sludge accumulation at a plant is a random process (Larson, 1988), and the design of “hot meals” delivery system (Bartholdi *et al.*, 1983). Part I of this research will focus on the VRPSD.

1.2 Introduction to the Generalized Traveling Salesman Problem (GTSP)

The classical traveling salesman problem (TSP) is well studied in the literature (Lawler *et al.*, 1985; Aarts and Lenstra, 1997; Korte and Vygen, 2000). The problem has many applications, such as large-scale integration (VLSI) chip fabrication (Korte, 1989), X-ray crystallography (Bland and Shallcross, 1989), etc.

In the classical TSP, each node must be visited exactly once. Nevertheless, this constraint is not always necessary and can be relaxed in some situations, where one only needs to visit a subset of the customers. The problem becomes a GTSP: firstly to find a proper subset of customers, and secondly to find the optimal visiting order in the selected subset.

The essence of the GTSP is to select a subset of the customers for visiting. In Mittenthal and Noon (1992), the GTSP is called a traveling salesman subset-tour problem (TSSP). To characterize a desired trait of an optimal subset-tour, the TSSP usually appears in applications with an additional constraint. This is the reason why Mittenthal and Noon (1992) called it the TSSP+1 class problem. Corresponding to different constraints imposed, several types of the problem are studied in the literature. Some representative examples include the prize collecting traveling salesman problem (Balas, 1989; Balas, 1995) and the time constrained TSP (Cloonan, 1966) or orienteering problem (Golden *et al.*, 1987).

The prize collecting traveling salesman (PCTS) problem was firstly introduced by Balas and Martin (Balas and Martin, 1991; Balas, 1995). The problem was formulated as a model for scheduling the daily operation of a steel rolling mill. Associated with each customer, in addition to the profit of visiting it, there is a penalty if the salesman fails to visit it. The objective is to minimize the travel costs and the net penalties, while satisfying the constraint that enough cities are visited to collect a prescribed amount of prize money.

Different from the PCTS problem, the objective of the time constrained TSP (TCTSP) is to maximize the profit realized from serving a subset of customers subject to the time constraint imposed on the problem. This problem was first introduced and discussed by Cloonan (1966). Some researchers also call TCTSP the selective traveling salesman problem (STSP) where they consider a preset constant route length as the constraint, see Laporte and Martello (1990) and Gendreau *et al.* (1998a, 1998b).

The orienteering problem (OP) only differs from the TCTSP in that the start point and the end point may not be the same. The name "orienteering problem" was originated from an outdoor sport: orienteering. Golden *et al.* (1987) provided its definition, and employed it to model and solve the problem of delivering home heating oil.

Among the three types of GTSP discussed above, the TCTSP (or the OP) is closely related to the problem considered in Part II of this study, where it is firstly extended to a multi-period TCTSP, then extended to a stochastic TCTSP.

1.3 Scope and Purpose of this Study

The scope of this research consists of the following two main parts.

1.3.1 Scope and Purpose of Part I of this Study

Part I focuses on the VRPSD. As the most studied problem among the SVRP, there are a number of algorithms available for solving VRPSD under the solution framework of *a priori* optimization. However, different researchers made various assumptions on the problem data in the literature; therefore, the performances of the algorithms proposed were evaluated based on different assumptions. In such cases, it is very difficult for a decision maker to know which algorithm is more preferable in a certain context. Therefore, firstly in Part I of this study, we try to carry out a comparative study on the representative algorithms for solving VRPSD, so that suggestions and recommendations can be made available for the practitioners in various contexts.

Most of the heuristics proposed for VRPSD in the literature are based on classical local search algorithms. One drawback of the classical local search algorithm is the tendency to be easily trapped in a local optimal solution. Due to the feature that metaheuristics can accept deteriorations in objective function value to some extent, it has the ability to escape from the local optimum and therefore may get global optimal solution. Thus, secondly in Part I of this study, we try to examine how modern metaheuristics behave for the VRPSD.

The contribution of this part of the study is twofold. Firstly, by carrying out the comparative study, we can determine which algorithm is superior to the others in a certain context. Therefore, some valuable suggestions can be provided for the practitioners. Secondly, we propose three metaheuristics, the simulated annealing (SA), threshold accepting (TA), and tabu search (TS) algorithms for the VRPSD. By comparing the performance of the proposed metaheuristics with that of the heuristics presented in the literature in various situations with respect to problem size and demand pattern, we can determine whether metaheuristics are suitable for solving this kind of problems, and also determine which metaheuristic is superior to the others in a certain context. Therefore, we can provide more choices and more valuable suggestions to the practitioners.

1.3.2 Scope and Purpose of Part II of this Study

The time constrained TSP (TCTSP) is the main theme of Part II of this thesis. The problem firstly considered in Part II of this study is a multi-period TCTSP in a rolling schedule environment, which can be frequently encountered in the practice. Consider a company providing services to the customers. A customer calls for service by specifying a desirable period and a time tolerance. Of course, the time tolerance can be zero, which means that the service is urgent and if the company can not provide service at the specified period, the customer would resort to other companies. With the presence of the time tolerance, the company can develop more flexible and more profitable schedules by considering the proximity of the customers requiring services, and considering the number of customers requiring services in different periods. In the former case, suppose that a customer j requires service in period i_1, and it can also be visited in period i_2, if the customers can be visited in period i_2 are in closer proximity to customer j than those requiring services in period i_1, it may be more profitable to schedule customer j in period i_2. In the later case, if the number of customers require services in different periods is very lumpy, delaying or bringing forward the service of some customers may be more profitable. This gives rise to the multi-period TCTSP: construct a schedule consisting of several periods rather than one period, find a subset of customers as well as the sequence of serving them in each period, so that the average profit per period in the long run is maximized.

The contribution of this part of the study can be summarized as follows. Firstly, from the aspect of theoretical study, the multi-period TCTSP is seldom studied in the literature, though it can be frequently encountered in the practice as described above. We provide a systematic study of this problem in this thesis: 1) We incorporate the concept of rolling schedule into the study of the problem due to the dynamic nature of the customer information. 2) We present a set-covering type formulation of the problem within one rolling horizon. Therefore, with the elongated rolling horizon and some assumptions regarding the customer demand information, an upper bound for this problem can be found by the column generation method. This type of formulation and the column generation solution method can be applied to similar problems, such as the team orienteering problem (Chao *et al*. 1996b), to find the optimal or an upper bound of the problems. 3) We provide several efficient heuristic methods with good performance in terms of both solution quality and computational time for this kind of problem. Moreover, the heuristics are studied against the upper bound and against each other under different problem parameter settings, so that the performance of each heuristic is clear under different scenarios. Secondly, from the aspect of practical application, based on the evaluation and comparison of the performance of the heuristics, suggestions and recommendations in different scenarios can be made for potential applications and therefore provide a guideline for the decision makers in their decision process.

The second problem studied in Part II of this study is the TCTSP with stochastic travel and service times. In the TCTSP, due to the effective working time limit constraint, one factor directly affects the total profit generated from the TCTSP tour is the travel and service time required for visiting the customers, which is usually assumed to be deterministic. However, in practical situations, both travel time and service time are not likely to be known with certainty in advance. The weather conditions (rain or snow) and the traffic conditions (road repair or traffic accidents) may impact on the travel time between the customers; while the service time is usually determined by the kind of service a customer requires. Obviously, the travel and service time is very important in the TCTSP, and it will directly affect the solution and therefore the profits generated from the solution. However, the stochastic nature of the problem never studied in the literature for this problem. Therefore, secondly in Part II of this thesis, we try to present models and solution methods for the stochastic TCTSP: the TCTSP with stochastic travel and service times.

1.4 Structure of the Thesis

Corresponding to the two types of the problems considered in this study: the vehicle routing problem with stochastic demands (VRPSD), and the time constrained traveling salesman problem (TCTSP), this thesis is mainly divided into two parts. Part

I covers Chapter 3 and Chapter 4. Part II includes Chapter 5 and Chapter 6. Chapter 2 provides a literature review on the solution frameworks and algorithms for the SVRP and the GTSP. The last chapter, Chapter 7, summarizes some conclusions for the whole thesis and directions of further research.

In Chapter 3, a comprehensive comparative study is carried out among three algorithms presented in the literature for the VRPSD. By building a common ground for comparison and making some adaptations to the original algorithms, the comparative study examines how the algorithms perform in various situations (with the increase of problem size, demand mean and/or variance, etc) under the assumption that demands follow both uniform and normal distributions. The comparative study also investigates whether the algorithms are sensitive to demand distribution type. In Chapter 4, several metaheuristics are presented for VRPSD, which include simulated annealing (SA), threshold accepting (TA), and tabu search (TS), etc. Computational results from these metaheuristics are compared with results from other algorithms presented in the literature; suggestions and recommendations are made for the potential applications in various scenarios. Chapter 5 focuses on the multi-period TCTSP in a rolling schedule environment. Heuristic methods based on iterative customer assignment and iterative center-of-gravity are developed for the multi-period TCTSP. To study the performance of these heuristics, we formulate the multi-period TCTSP as a set-covering problem, and propose a column generation scheme to solve its linear programming (LP) relaxation to optimality to get an upper bound for the original problem. In Chapter 6, we consider the TCTSP in the stochastic case, where the travel and service times of the problem may become random variables. Models formulated as both chance-constrained program and stochastic program with recourse are provided, and an integer L-shaped solution method is proposed for solving it.

Chapter 2

Literature Review

This chapter summarizes research work that has been done in the literature for the stochastic vehicle routing problem (SVRP) and the generalized traveling salesman problem (GTSP). Section 2.1 covers literature for the various types of the SVRP. Section 2.2 focuses on one type of SVRP, the VRPSD. The solution framework, recourse policies and algorithms available for the VRPSD are discussed in detail in this section. Literature on several types of the GTSP is presented in Section 2.3. Finally, Section 2.4 summarizes some findings in the literature review and their relationship with the following chapters of the thesis.

2.1 General Overview of the Literature on SVRP

The SVRP addresses the problem of constructing vehicle routes through the customers that minimizes the expected distance traveled with the presence of uncertainty of some problem parameters. Though comparing with their deterministic counterparts, relatively less efforts and achievements have been made on the SVRP, there is still much literature available for various types of SVRP.

(1) The probabilistic traveling salesman problem (PTSP)

When a postman delivers mails to the customers, obviously, he does not expect each customer needs a visit each day. When the customer presence is a random variable, and is described by a probability p_i, the problem of finding a least expected cost cycle becomes the PTSP. This problem was introduced by Jaillet (1985, 1988). The author derived closed form expressions to obtain efficiently (in polynomial time of low order) the expected length of tours under various probabilistic assumptions. By analyzing the closed form expressions, some properties and characteristics of optimal solutions to PTSP were derived. The paper also presented the specific conditions under which the TSP solution can serve as a good approximation for the PTSP. However, their results show that, in general, entirely new solution procedures are necessary to devise for PTSP. Bertsimas *et al.* (1990) also addressed the PTSP. They discussed the applicability of *a priori* optimization strategies. They showed that if the nodes are randomly distributed in the plane, the *a priori* strategies behave asymptotically equally well on average with re-optimization strategies. Two kinds of heuristics using the *a priori* strategies were also presented in the paper. The first is based on the space-filling curve heuristic, while the second is based on methods seeking local optimality, which includes 2-opt, 3-opt, 1-shift, etc. In Laporte *et al.* (1994), the authors formulated the PTSP as an integer linear stochastic program. Under the *a priori* strategies, the authors presented the first exact algorithm for this kind of problem. The algorithm is based on a branch-and-cut approach, which relaxes some of the constraints and uses lower bounding functionals on the objective function.

(2) The traveling salesman problem with stochastic traveling times (TSPST)

Among the problem parameters: customer demand, customer presence and travel time, etc., travel time is the parameter that most unlikely to be known for sure in advance (while constructing the routes), due to the weather and traffic conditions. However, the TSPST is less studied compared to other SVRP in the literature. In Kao (1978), under the assumption that the probability of a sum of random travel times can be readily computed, two heuristics for this problem were proposed: one is based on dynamic programming; and the other employs the implicit enumeration to find a solution. In Sniedovich (1981), the author pointed out that, the monotonicity property required by the dynamic programming algorithm was not verified in Kao (1978); therefore the algorithm may obtain sub-optimal solutions. This difficulty was overcome in Carraway *et al.* (1989), where a generalized dynamic programming algorithm was proposed and applied to TSPST. Another version of TSPST is *m*-TSPST, where *m* vehicle routes all start and end at a common depot. Lambert *et al.* (1993) designed the money collection routes through bank branches in the case of stochastic traveling times, due to the fact that congestion of some arcs usually happens in the rush hour. To take the stochastic traveling times into consideration, the objective function includes two penalty terms: one is due to the fact that money accumulated between vehicle arrival time and a branch's closing time is not collected until the next day, therefore it is preferable to delay as much as possible visits to branches; the other is due to the fact that all money contained in the vehicles arriving at depot later than a prescribed time loses one day's interest. The authors applied the adapted Clark and Wright (1964) algorithm to solve the VRP with stochastic traveling times. In addition to the stochastic traveling times, Laporte *et al.* (1992) considered stochastic service times at the vertices as well. Here the penalty for late arrival is proportional to the length of the delay. Three mathematical programming models were presented in the paper, a chance-constrained model, a three-index simple recourse model, and a two-index recourse model. The paper also presented a general branch-and-cut algorithm for solving the three models.

(3) The vehicle routing problem with stochastic customers (VRPSC)

In the PTSP, vehicle capacity constraint is relaxed. When a customer is present with probability p_i but with deterministic demand, and the vehicle capacity constraint must be respected, the PTSP is extended to VRPSC. The best source of theoretical information on VRPSC is Bertsimas (1988), in which several properties, bounds and heuristics for the problem were described. Benton and Rossetti (1992) considered general demands and proposed an empirical comparison of three operating policies: follow the planned route without skipping absent customers (fixed route), skip absent customers (modified fixed route), and re-optimize the remaining route whenever the absence of a customer is revealed (variable route). The author assumed that demands are known at the beginning of the period in which they occur, so it is possible to modify the fixed route or reschedule the fixed route whenever the absence of customers are known. For the fixed route alternative, by using the expected value of non-zero demands, the total cost can be solved by classic VRP heuristics. However, because of the randomness of customer presence, the total cost of the other two alternatives must be calculated for each period. The cost of modified fixed-route alternative can be solved by skipping the appropriate zero demand customers from the VRP solution. The cost of variable route alternative is solved by applying an efficient heuristic VRP procedure to the customers with non-zero demand for that period. Finally, the one with the least total cost in each period is chosen as the best alternative. Waters (1989) also applied the above-mentioned three alternatives to deal with VRP with stochastic customers, but from a different point of view. In practice, the third alternative of variable routes is not always possible, because the customers to be omitted must be known some time before vehicles set out, to allow time to produce new routes. Therefore, the problem the paper studied is: how large are potential savings of using modified-fixed routes and variable routes, its relationship with the number of absent customers, and the break-even points (the proportion of absent customers) to make rescheduling worthwhile over fixed and modified-fixed routes.

(4) The vehicle routing problem with stochastic demands (VRPSD)

In VRPSD, the customer demand is a random variable while all the other problem parameters are assumed to be deterministic. As VRPSD is the most studied SVRP in the literature and it is the focus of Part I of this study, Section 2.2 will present a more detailed literature review on the solution framework, recourse policies and the algorithms available for VRPSD.

(5) The vehicle routing problem with stochastic customers and demands (VRPSCD)

As a combination of the VRPSC and VRPSD, VRPSCD is an extremely difficult problem (Gendreau *et al.* 1996b). Bertsimas (1992) presented the closed-form recursive expressions and algorithms to compute the expected length of an *a priori* sequence under general probabilistic assumptions. Also the upper and lower bounds on the *a priori* and re-optimization strategies were derived for this kind of problems. The purpose is to compare these strategies from a worst and average case perspective. Heuristics based on cyclic heuristic (Haimovitch and Rinnooy Kan, 1985), were proposed and their worst-case performance as well as their average behavior were analyzed in the paper. Gendreau *et al.* (1996a) presented a tabu search algorithm for this problem. Based on an initial solution constructed by Clark and Wright (1964) algorithm, the neighborhood of a solution X contains all solutions that can be reached by removing in turn one of *neighbor_size* randomly selected customers, and inserting each of them either immediately before, or immediately after one of its ϖ nearest neighbors. If a vertex is moved from route π to the same route or to a different route at iteration i, its reinsertion or displacement in route π is tabu until iteration i + *NoTabu*, where *NoTabu* is the tabu tenure and is randomly selected in the interval [*n*-5, *n*]. However, the penalized objective function can not be used directly to evaluate the moves and select the best move for the tabu search, due to the computational burden in the case of stochastic customers and demands. One of the major contributions of the paper is the development of an easily computed proxy for the objective function, to be used in the evaluation of potential moves, and also the elaboration of a series of mechanisms aimed at efficiently managing the proxy. Ong *et al.* (1997) provided a framework to model customers in a due-date environment. In addition to the stochastic demand, each customer requires a service on a specific day (due-date) and at a particular time window of the day. In the objective function of their model, in addition to the routing cost, there are two penalty terms: one is associated with the overdue dissatisfaction of each customer and the expected losses of the company; the other is related to the customers that can not be served fully on the planned route. The paper presented a "LOSS function" based on due-date to serve as selection criteria of customers to be served. The stochastic demand was handled based on the chance-constrained model (Stewart and Golden, 1983). To take the time window constraint into consideration, the paper proposed an adaptation of the insertion heuristic by Solomon (1987) for the routing and scheduling. Gendreau *et al.* (1995) presented an exact algorithm for this problem, which used an Integer L-Shaped method. Solutions were reported for instances involving up to 46 vertices solved to optimality.

2.2 Literature Review on Recourse Policies and Algorithms for VRPSD

Since VRPSD is the most studied problem among the SVRP and it is the focus of Part I of this study, the solution framework, recourse policies and available algorithms are discussed in detail as follows.

2.2.1 Solution Concepts and Recourse Policies

For the vehicle routing problem with stochastic demands (VRPSD), solution frameworks mainly depend on the operating policies (whether re-optimization is allowed) adopted and the time when demand information is available. Two solution frameworks are available in the literature: stochastic programming and Markov decision processes. The former belongs to the *a priori* or static method, because the order of the customers' visitation is not changed during its real time execution; while the later belongs to real time or dynamic method, because routes are recomputed based on the information that becomes available during the execution of the tour. An inherently dynamic formulation was proposed by Dror *et al.* (1989). They developed a Markov decision process model for the VRPSD, but no computational experience was provided. Dror (1993) studied a slightly modified version of the model, also no computational experience was provided and the author considered instances with more than three customers as computationally intractable. Secomandi (1998) proposed different Markov decision process models for VRPSD solved in the dynamic context. Moreover, the author developed an exact dynamic programming algorithm to compute a dynamic optimal policy; he also proposed a heuristic dynamic programming algorithm to compute a partially dynamic policy and an on-line rollout algorithm to compute a dynamic routing policy.

However, the dynamic routing policy may be impractical or even impossible in practical applications due to the following reasons:

1) Not enough resources to repeat the redesigns;
2) Not enough information regarding demands before actually visiting the customer, etc.

Therefore, one representative method in the literature is to determine a fixed *a priori* sequence among all potential customers, and consider recourse actions upon a route failure. The idea of using *a priori* sequence was first proposed for the PTSP in Jaillet (1985). Bertsimas (1988) generalized the idea and applied it to other combinatorial optimization problems, such as the probabilistic minimum spanning tree problem, the PTSP, the probabilistic vehicle routing problem, and facility location problems. All studies above assume that the demand distribution is binary, i.e., customer i either has 1 unit demand with probability p_i, or does not have any demand with probability 1 - p_i. The idea is further generalized to the arbitrary discrete-demand distributions in Bertsimas (1992).

Within the framework of the *a priori* optimization method, the VRPSD can be formulated both as a chance-constrained program (CCP) and as a stochastic program with recourse (SPR). In chance-constrained program, one seeks a first stage solution for which the probability that all demands on a route exceeding the vehicle capacity is not greater than a predefined probability level. Under this condition, no recourse action is adopted in case of route failure. Under some assumptions, a chance-constrained model can be transformed into an equivalent VRP with an artificial vehicle capacity. Therefore existing algorithms for VRP can be applied to the resulting problem in this case.

In the stochastic program with recourse, the problem is solved in two stages. In the first stage, the objective is to determine a solution that minimizes the expected cost of the second stage solution. Specifically, in the first stage, a planned or *a priori* solution is determined. In the second stage, as the actual demands are revealed, the first stage solution may not be possible as planned because of the route failure, for example, the total demand of a route may exceed the vehicle capacity. A recourse or corrective action is then applied to the first stage solution. The total expected distance traveled includes two parts: one is the fixed length of the *a priori* sequence; the second is the expected value of the additional distance traveled whenever demand on the sequence exceeds vehicle capacity.

For a given VRPSD, two categories of recourse approaches can be found in the literature. One recourse approach belongs to the dynamic category, which re-optimizes the remaining portion of the route upon each failure based on the information that becomes available during the execution of the tour. Among those static recourse policies, a simple and obvious one is that, whenever route failure occurs, go back to depot to restock. In the two SPR models presented in Stewart and Golden (1983), one applies a penalty proportional to the probability of exceeding the vehicle capacity, the other uses a penalty proportional to the expected demand in excess of the vehicle capacity. Both Bertsimas *et al.* (1995) and Yang *et al.* (2000) employed a dynamic programming procedure to plan "preventive breaks" at strategic points along the first stage route, rather than waiting for route failure to occur. The difference is that, in Yang *et al.* (2000), partial delivery is permitted, though penalized by imposing a fixed nonnegative cost whenever route failure occurs. These recourse policies, though different from one another, belong to the static approach, because the order of the customers' visitation is not changed during its real time execution. In this study, we will focus on the fixed *a priori* static method.

2.2.2 Available Algorithms for VRPSD in the Literature

Exact algorithms for the SVRP are developed based on mixed or pure integer stochastic programs; see Laport *et al.* (1992, 1994) and Gendreau *et al.* (1995). The integer L-shaped method was employed to solve the SVRP in the above papers. It is an extension of the L-shaped method of Van Slyke and Wets (1969) for solving the two stage stochastic linear problems when the random variables have finite support, by incorporating a branching procedure to recover the integrality of the variables. As a branch-and-cut algorithm applicable to a wide range of stochastic integer programs with recourse, the integer L-shaped method has also been applied in solving the VRPSD. Hjorring and Holt (1999) derived more effective optimality cuts and a tight global lower bound on the second stage value function based on the concept of partial routes for the single vehicle case. Laporte *et al.* (2002) studied lower bound on the second stage value function for the normal and Poisson distributed demands. They also constructed their optimality cuts based on the concept of partial routes in Hjorring and Holt (1999). In addition, Dror *et al.* (1993) considered the VRPSD, in which the number of potential failures per route is restricted either by the data or the problem constraints. A chance-constrained version of the problem was considered and solved to optimality by algorithms similar to those developed for the deterministic VRP. Then three classes of recourse models were analyzed. Under the assumption that route failure can only occur at most once, an exact solution with a very high probability of being optimal was easily computed by solving a sequence of deterministic problems.

The VRP is a combinatorial NP-Hard problem (Bodin *et al.*, 1983). By adding the stochastic element to the demands, the problem becomes even more difficult to solve in terms of computational time as intricate probability computations are usually involved. Known approaches for solving these problems optimally suffer from an exponential growth in computation time with problem size, which is very unlikely to be acceptable in the real world. Therefore, considerable attention and research efforts have been devoted to the development of efficient heuristics (approximate algorithms) to get near optimal solutions for large sized problems.

The first heuristic for the VRPSD was proposed in Tillman (1969), which is for multiple depot case and the algorithm is based on Clark and Wright (1964). In Stewart and Golden (1983), in addition to presenting one CCP and two SPR models, they also considered several demand distributions and proposed two heuristics: one based on Clark and Wright (1964), the other based on Lagrangean relaxation.

Bertsimas *et al.* (1995) presented an *a priori* heuristic based on the cyclic heuristic (tour construction), 2-interchange and the dynamic programming (tour improvement). Computational results were presented based on two types of demand distributions: discrete uniform distribution and discretised normal distribution. They considered the single vehicle case, because in VRPSD, returning trips to the depot are permitted, and therefore vehicle capacity becomes a soft constraint. Moreover, Yang *et al.* (2000) shown that, with the presence of no additional constraints, it is not necessary to use multiple vehicles due to the recourse policy, the optimal route is always a single one. Nevertheless, with the presence of such constraints as a limit on the maximum traveling distance or effective working time of a vehicle, a single route may not be usable in most real-world situations. Therefore, in Yang *et al.* (2000), they proposed heuristics for both the single vehicle and multiple vehicle cases. In the single vehicle case, a composite method (insertion + Or-opt) was used to build a single route. For the multiple vehicle case, they applied classic route-first-cluster-second and cluster-first-route-second heuristics to solve this problem, under the assumption that the expected route length of each route must be within a predefined limit. In their computational experience, customer demands were assumed to follow discretised triangular distribution.

In addition to the traditional heuristics discussed above, modern heuristic, such as simulated annealing (SA), has also found its application in solving VRPSD. Teodorovic and Pavkovic (1992) presented a SA algorithm, which is limited to the situation where at most one route failure occurs in each route. Under this assumption, they first introduced how to calculate the expected cost; then presented a two-stage scheme, both of which utilize SA algorithm, with the first stage SA serving as a clustering procedure and the second stage SA serving as a routing procedure. The computational results were presented based on uniformly distributed customer demand information.

2.3 Literature Review on the Generalized Traveling Salesman Problem

As a relaxed variant of the TSP, where not each customer is required to be visited exactly once, the GTSP tries to select a subset of the customers with a desired trait which is usually described as an additional constraint imposed on the subset tour. The GTSP has received increasing attention in recent years. Most studies focus on the prize collecting traveling salesman (PCTS) problem, the time constrained TSP (TCTSP) and the orienteering problem (OP).

For the PCTS problem, Balas (1989, 1995) presented an intensive theoretical study. In Balas (1989), he discussed the structural properties of the PCTS polytope, the convex hull of solutions to the PCTS problem. In particular, he identified several families of facet defining inequalities for this polytope, which can be used in developing algorithms for the PCTS problem either as cutting planes or as ingredients of a Lagrangean optimand. In Balas (1995), he presented a general method for deriving a facet defining inequality for the PCTS polytope from any facet defining inequality for the asymmetric traveling salesman (ATS) polytope. The method was applied to several well-known families of facet inducing inequalities for the ATS polytope. The cloning and clique lifting procedure for the ATS polytope was also extended to the PCTS polytope in his paper. In addition to the theoretical study, a number of heuristics have been developed for the PCTS problem and its several variants. In Bienstock *et al.* (1993), they considered a simplified version of PCTS problem, where the objective is to find a tour that visits a subset of the vertices such that the length of the tour plus the sum of penalties of all vertices not in the tour is as small as possible. They presented an approximation algorithm with constant bound. The algorithm is based on an algorithm presented in Christofides (1976) for the TSP as well as a method to round fractional solutions of a linear programming (LP) relaxation to integers, feasible for the original problem. In Lopez *et al.* (1998), they considered the hot strip mill production scheduling problem for scheduling steel coil production in the steel industry. The problem was modeled as a generalization of the PCTS problem with multiple and conflicting objectives and constraints. They presented a heuristic based on tabu search and a new idea of “cannibalization” for solving the problem. In Awerbuch *et al.*

(1998), they presented the first approximation algorithm having a polylogarithmic performance guarantee for the PCTS problem and two variations of the problem. In one variation, there is no penalty associated with each unvisited city. Another variation is called the “orienteering problem" by Golden *et al.* (1987).

The TCTSP was first introduced and discussed by Cloonan (1966). Gensch (1978) proposed a solution method based on tree search for an industrial application of this problem. The algorithm requires finding both the lower bound and the upper bound of the solution to get an optimal sub-tour for the problem. The lower bound is found by constructing a sub-tour by the nearest neighbor heuristic; while the upper bound is obtained by solving a time constrained assignment problem by relaxing the sub-tour elimination constraints, which applied the generalized Lagrange multiplier method. Golden *et al.* (1981) developed a simple iterative procedure for the problem. When the constraint considered is a preset constant route length rather than a time limit, some researchers call TCTSP the selective traveling salesman problem (STSP). Laporte and Martello (1990) proposed an exact algorithm, which consists of embedding a LP relaxation within a branch-and-bound scheme. Another exact algorithm proposed by Gendreau *et al.* (1998a) is based on branch-and-cut procedure. Heuristics for STSP can be found in Laporte and Martello (1990) and Gendreau *et al.* (1998b).

The definition of OP was first provided in Golden *et al.* (1987). Ramesh *et al.* (1992) presented an exact algorithm for OP. In Leifer and Rosenwein (1994), they tightened the LP relaxation and proposed a procedure to obtain upper bounds. In Kataoka *et al.* (1998), they proposed a minimum directed 1-subtree problem as a relaxation to the OP and developed a cut and dual simplex method and a Lagrangean relaxation method to improve its lower bound. Since the OP is NP-hard (Golden *et al.*, 1987), a number of heuristics have been developed during the past few years. Tsiligirides (1984) also studied this problem; however, he called it the generalized TSP. Two heuristics were presented in Tsiligirides (1984): one is deterministic, and the other is stochastic. In the deterministic heuristic, the geographic region is divided into sectors by concentric circles. Routes are then built within sectors to minimize the total travel time. The stochastic algorithm selects the best route among a large number of routes created based on Mont Carlo techniques. When constructing the OP tour, to select a node *j* for inclusion, he applied a measure of desirability for all nodes not currently on the route. The largest 4 values of the measure of desirability are selected and normalized so that they sum to one. A random number is then generated according to U (0, 1) in order to select a node *j* for inclusion. The procedure is repeated until no more nodes can be included into the route. Since the inclusion of each node *j* is randomly selected, many routes can be generated in this method. An algorithm based on the concept of center-of-gravity was proposed by Golden *et al.* (1987). The heuristic includes three steps: route construction, route improvement and center-of-gravity step. After constructing an initial route, they applied 2-opt to improve the route, followed by a cheapest insertion step to insert as many nodes as possible to the initial route without violating the constraint. Suppose that the route resulted from the first two steps is called route π. In the center-of-gravity step, they first calculate the center-of-gravity for route π by using the coordinates and profits of all nodes in route π. Then, for each node *i*, calculate the ratio of its profit to the traveling time from node *i* to the center-of-gravity of route π. Add nodes to the route in descending order of this ratio using cheapest insertion, until no additional nodes can be added without violating the time limit constraint. The route improvement step is then applied again to make some adjustments to the resulting route. After getting the new route π', the center-of-gravity step is applied again to form a new route π''. The procedure is repeated until a cycle develops, and then the route with the highest score is selected as the final solution. In their computational experience, Golden *et al.* (1987) claimed that their heuristic outperforms the two heuristics proposed in Tsiligirides (1984). An improved algorithm that incorporates the center-of-gravity idea, Tsiligirides's randomization concept, along with learning capabilities was presented by Golden *et al.* (1988). In Keller (1989), he adapted the algorithm originally developed for the multiobjective vending problem (Keller, 1985) to solve the OP, and made a comparison with algorithms from Tsiligirides (1984) and Golden *et al.* (1987). In Mittenthal and Noon (1992), they presented a heuristic for the TSSP+1 class of problems, which explores the solution space by either insertion of unvisited vertices or deletion of included vertices from the subset tour. A fast and effective heuristic that consists of one initialization step and one improvement step was presented by Chao *et al.* (1996a). A similar procedure was employed to address the team orienteering problem in Chao *et al.* (1996b). Moreover, Kantor and Rosenwein (1992) addressed the orienteering problem with time windows. They developed a tree heuristic based on an exhaustive search of the feasible solution space.

2.4 Conclusion and Further Remarks

From literature presented for the SVRP (Sections 2.1 and 2.2), it is clear that VRPSD is the most studied among various types of SVRP in terms of solution concepts, recourse policies and algorithms proposed in the literature. Moreover, most of the researchers formulated VRPSD as a stochastic program with recourse within the framework of the *a priori* optimization. Several representative heuristics for solving

VRPSD include Bertsimas *et al.* (1995), Yang *et al.* (2000) and Teodorovic and Pavkovic (1992). However, different researchers made different assumptions on the problem data when doing the computational experiment; and the performances of the algorithms proposed were evaluated based on their assumptions. For example, in Bertsimas *et al.* (1995), the demands are assumed to follow normal and uniform distributions; in Yang *et al.* (2000), the demands follow triangular distribution; in Teodorovic and Pavkovic (1992), the demands follow uniform distribution, and they also assumed that at most one route failure could occur for each route. In such cases, it is very difficult for a decision maker to know which heuristic is more preferable in a certain context. Therefore, in Chapter 3 of this thesis, a comprehensive comparative study is carried out on algorithms presented in Bertsimas *et al.* (1995), Yang *et al.* (2000) and Teodorovic and Pavkovic (1992), so that suggestions and recommendations can be made available for the practitioners in various contexts.

One other finding from the literature review of SVRP is that, most of the heuristics proposed for VRPSD in the literature are based on classical local search algorithms (except for the SA algorithm presented in Teodorovic and Pavkovic, 1992), which have the tendency to be easily trapped in a local optimal solution. Though there is an SA algorithm presented in Teodorovic and Pavkovic (1992), results from our comparative study (Teng *et al.*, 2001) show that SA algorithm based on the original neighborhood generation method and cooling schedule performed quite badly in comparison with the other heuristics with respect to both solution quality and computational time. Also, it seemed that there are some randomness involved in both the solution quality and computational time. This does not mean that SA algorithm is not suitable for VRPSD, because the performance of the SA algorithm depends greatly on the neighborhood generation mechanism and the cooling schedule. Thus, in Chapter

4 of this thesis, we present three metaheuristics, simulated annealing (SA), threshold accepting (TA) and tabu search (TS) for solving VRPSD, and try to examine how modern heuristics behave for the VRPSD.

Literature presented for the GTSP in Section 2.3 indicates that, the TCTSP and OP are most closely related to the problems considered in Part II of this study. For TCTSP and OP, except for the team orienteering problem considered in Chao *et al.* (1996b), most work done in the literature is to construct a single orienteering tour (TCTSP tour) in an optimal way or through heuristics. Chapter 5 of this thesis extends the TCTSP into a multi-period TCTSP, and presents some heuristics for solving it in a rolling schedule environment.

In the literature as well as in the multi-period TCTSP considered in Chapter 5, both the travel time and the service time are assumed to be deterministic, though, in practical situations, both are not likely to be known with certainty in advance. Nevertheless, the profit generated from the TCTSP is directly affected by the travel and service times, due to the time limit constraint of the TCTSP. To make the problem considered more sensible, in Chapter 6 of this thesis, the TCTSP is extended to the stochastic case, where the travel time and the service time are assumed to be random variables.

Chapter 3

Comparative Study of Algorithms for VRPSD

For SVRP, most studies have been made on the VRPSD and several heuristics can be found in the literature. In this chapter, we compare the performance of three representative heuristics from the literature based on the model of stochastic program with recourse (SPR). A common ground for the comparative study is built first, followed by conducting a comprehensive computational study on each of the three selected heuristics. We examine how the heuristics perform in various situations with respect to problem size and demand pattern, in order to determine which algorithm is superior to the others in a certain context. Based on these findings, some valuable suggestions are recommended to the practitioners.

3.1 Problem Statement

3.1.1 Problem Description

The vehicle routing problem with stochastic demands (VRPSD) can be described as follows: Given a set $V = \{1, \ldots, n\}$, and $V' = V \cup \{0\}$, with 0 representing the depot, and 1, … ,n denoting the n customers, a distance matrix $C = (c_{i,j})$, $i \in V'$, $j \in V'$, is defined on Euclidean plane. There is a fleet of m identical vehicles of capacity Q at the depot that deliver goods to the n customers. Let D_i, $i = 1, 2, \ldots, n$, denote the random variables that describe the demand of customers. We assume that all D_i are independent random variables following a known discrete probability distribution with

K+1 possible values, 0, 1, 2, … , K. Let $p_i(k)$ be $P(D_i = k)$, the probability that demand at node i is k. We further assume that $K < Q$; that is, the vehicle's capacity is greater than the largest demand of a customer in a given period. This assumption removes the consideration of multiple returns to the depot from the same node. Moreover, we assume that L is a predefined maximum limit for the expected route length. The VRPSD tries to determine vehicle routes, restocking policy at each node, as well as the number of vehicles m, under the following conditions:

(i) All routes start and end at the depot;

(ii) Each city is visited exactly once except the depot;

(iii) The expected distance traveled by any vehicle does not exceed the prescribed value L;

(iv) The total expected distance traveled by all vehicles is to be minimized.

Note that, though we evaluate the routes by using the expected route length, we also call it the expected cost in the following parts of this thesis.

3.1.2 Calculation of the Expected Cost

In Bertsimas *et al.* (1995), they presented the following equation to calculate the expected cost of each tour:

$$E[L_\tau] = \sum_{i=0}^{n}(1-\gamma_i)c_{i,i+1} + \sum_{i=1}^{n}(2\delta_i c_{i,0} + \gamma_i[c_{i,0} + c_{i+1,0}]) \qquad (3.1)$$

where γ_i is the probability that the demand at the i^{th} node is exactly equal to the stock available in the vehicle, δ_i is the probability that the demand at the i^{th} node exceeds the remaining stock, and $c_{i,j}$ is the distance between nodes i and j in the tour. Bertsimas (1992) presented some equations to calculate γ_i and δ_i as follows:

$$\gamma_1 = 0$$

$$\gamma_i = \sum_{l=1}^{\lfloor iK/Q \rfloor} \left\{ \sum_{k=1}^{K} p_i(k)\, f(i-1, lQ-k) \right\}, \qquad 2 \le i \le n \tag{3.2}$$

$$\delta_1 = 0$$

$$\delta_i = \sum_{l=1}^{\lfloor iK/Q \rfloor} \left\{ \sum_{k=1}^{K-1} \left(\sum_{r=k+1}^{K} p_i(r) \right) f(i-1,\ lQ-k) \right\}, \qquad 2 \le i \le n \tag{3.3}$$

$f(s,r) = \Pr\{\text{the total demand of the customers } 1,\dots,s \text{ is } r\}$ is computed from the recursion:

$$f(s,r) = \sum_{k=0}^{\min\{K,r\}} p_s(k)\, f(s-1, r-k), \qquad s = 2,\dots,n,\ r = 0\dots Ks \tag{3.4}$$

with the initial conditions:

$$f(s,r) = 0 \qquad \text{for} \quad r > Ks \tag{3.5}$$

$$f(1,r) = \begin{cases} p_i(r) & \text{for} \quad 0 \le r \le K \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

3.1.3 Dynamic Programming (DP) Recourse Policy

Both Bertsimas *et al.* (1995) and Yang *et al.* (2000) employed the following dynamic programming (DP) recourse policy. Instead of waiting for route failure to occur, DP recourse policy can recursively decide at each stage (node), which is more preferable: go on visiting the next node, or return to the depot for restocking. A threshold value, vehicle remaining load for delivery operation, is calculated at each node *i*. If the remaining load after visiting node *i* is less than or equal to the threshold value, it is better to return to depot before visiting the rest of the customers.

For any given *a priori* solution to the VRPSD problem, let $(v_1, v_2, ..., v_n)$ be the set of nodes arranged in the same sequence as they appear in the visitation tour. Let *q* denote the vehicle's remaining load. Define $f_j(q)$ to be the expected remaining distance to travel, given that the optimal strategy is followed. The following dynamic programming recursion describes the decision about when the vehicle should return to the depot.

1) Initialization

   Set $f_n(q) = c_{0,v_n}, \forall q \in \{0, 1, ..., Q\}$ (3.7)

2) Recursion

   For $j = n$-1, …, 1: compute $f_j(q)$ = min { $f_j^1(q), f_j^2(q)$ }

   where

$$f_j^1(q) = c_{v_j, v_{j+1}} + \sum_{k:k \le q} f_{j+1}(q-k) p_{j+1}(k) + \sum_{k:k>q} [2c_{v_{j+1},0} + f_{j+1}(q+Q-k)] p_{j+1}(k) \quad (3.8)$$

$$f_j^2(q) = c_{0,v_j} + c_{0,v_{j+1}} + \sum_{k=0}^{K} f_{j+1}(Q-k) p_{j+1}(k) \quad (3.9)$$

3) Threshold calculation

$$q*(j) = \max\{q : f_j(q) = f_j^2(q)\} \quad (3.10)$$

Here, the terms $f_j^2(q)$ and $f_j^1(q)$ reflect the strategy of returning to depot, and the strategy of continuing to the next node, respectively.

The term $q^*(j)$ is the threshold value for node v_j. When the remaining load is very large, it is obvious that one should go on to the next node. As remaining load decreases, the cost of going on to the next node increases, the maximum remaining load $q^*(j)$ that makes going on to the next node more costly than going back to depot is the threshold value for node v_j. If the remaining load after visiting node v_j is less than $q^*(j)$, it is better to return to the depot before serving further demand points.

For each visitation sequence, through the backward dynamic programming, its expected cost can be calculated by the following equation.

$$z = \sum_{j=0}^{Q} p_{v_1}(j) f_1(j) + c_{0,v_1} \tag{3.11}$$

This expected cost, together with the threshold value at each node, forms a solution for any visitation sequence.

3.2 Review of the Selected Algorithms

In this section, we give a brief review of the algorithms selected for comparison in this study. Readers can refer to the original papers for details.

3.2.1 Bertsimas *et al.*'s Algorithm

Bertsimas *et al.* (1995) presented an *a priori* heuristic based on the cyclic heuristic, 2-interchange and dynamic programming (DP) procedure. They only considered the single vehicle case. This is possible in VRPSD, because returning trips to the depot are permitted, and vehicle capacity becomes a soft constraint.

Cyclic heuristic with local improvement and dynamic programming (DP)

For a given initial TSP tour (0, 1, 2, …, *n*), the cyclic heuristic generates *n* cyclic permutations, with the *i*^th^ permutation π_i = (0, *i*, *i*+1,…, *n*, 1, 2,…, *i*-1, 0). 2-interchange is then applied to make local improvement for each of the *n* orderings. The DP recursion along with the threshold calculation is used to determine when the vehicle should return to the depot. The heuristic then chooses the lowest cost visitation sequence and recourse policy as the final solution.

3.2.2 Yang *et al.*'s Algorithms

In Yang *et al.* (2000), a composite method (insertion + Or-opt) was proposed for single VRPSD. For the multiple VRPSD, they applied classic route-first-cluster-second (R1-C2) and cluster-first-route-second (C1-R2) heuristics to solve this problem. The calculation of the expected cost of the routes and the best restocking policy are performed by the DP recursion, which is similar to the recursion presented by Bertsimas *et al.* (1995). One difference is the recourse cost. To penalize a partial delivery, a fixed nonnegative cost is imposed whenever route failure occurs, in addition to the cost of traveling to the depot and back to the route. Additionally, to save the computational time, they present an approximation method to calculate the insertion cost.

Single vehicle routing problem

The heuristic first attempts to find a single route through all the customers using the insertion procedure (Bodin *et al.*, 1983). Next, Or-opt procedure (Or, 1976) is used to seek further improvements. In the Or-opt procedure, a set of *k* successive nodes are removed from the current location (I), and reinserted back into the route elsewhere other than where they are removed (II). The approximation method is used to calculate the cost improvement: the difference of cost savings (resulted from I) and the additional costs (resulted from II). The move with the maximum cost improvement is implemented. The procedure repeats until there is no cost improvement available and k is decreased by 1. The procedure terminates until k is 1 and there is no further cost improvement available.

Multiple vehicle routing - Route-first-cluster-second algorithm

The routing procedure is the same as the one used in the single vehicle routing. Suppose $\pi = (i_1, i_2, \dots i_n)$ is an ordered sequence obtained by using insertion and Or-opt algorithms. Given a limit L on the expected cost of each route, in the clustering procedure, Yang *et al.* (2000) proposed another dynamic programming procedure to partition the sequence into multiple vehicle routes. The procedure recursively determines the partitioning points for each subsequence such that the expected cost of the total route set is minimized and the expected cost of each subsequence is within the limit L.

Multiple Vehicle Routing - Cluster-first-route-second algorithm

In the clustering process, one needs to select the seed points first. Yang *et al.* (2000) employed the circle covering method of Savelsbergh and Goetschalckx (1995) to determine the seed points. Then, customer clusters are formed around the seed points in the following way: select the seed point not yet assigned with the smallest covering circle (see Savelsbergh and Goetschalckx, 1995), form a seed route with return trips between the seed point and the depot. Calculate the insertion cost of each unconnected node at the best location in the seed route. Successively insert the unconnected nodes with the least insertion cost into the route. Reject the nodes that make the route infeasible. The procedure proceeds until no more insertion is feasible; and other clusters are formed by repeating the same procedure.

In the routing process, they tried to reposition each node from its current position to any other position, either on the same route or to other routes, provided that the new position of the node does not violate the route length constraint.

Multiple vehicle routing - Improving the heuristic solution

To further improve the solution obtained from the above two heuristics, Yang *et al.* (2000) also repeatedly applied inter-route and intra-route exchange procedures until no further improvement can be made. The inter-route exchange tries to improve the routes by moving a segment of nodes from one route to another. The intra-route exchange procedure uses Or-opt to further improve each route.

3.2.3 Teodorovic and Pavkovic's Simulated Annealing (SA) Algorithm

Teodorovic and Pavkovic (1992) presented a SA algorithm to solve the problem in the situation where at most one route failure can occur in each route. They first introduced a way to calculate the expected cost in this case. The paper employs a two-stage scheme, both of which utilize SA algorithm. In the first stage, SA is used to find a satisfactory initial solution. The neighborhood solutions are generated by randomly constructing route sets. This procedure corresponds to the clustering of customers. In the second stage, selected customers are repeatedly exchanged only within each route.

The neighborhood is generated by swapping two nodes within each of the defined routes. It corresponds to the routing procedure.

3.3 Common Grounds for the Comparative Study

3.3.1 Criteria for the Measurement of the Comparative Study

The success of a heuristic for solving VRPSD depends on a number of aspects:

1) Performance, i.e. the running time and the quality of the final solution;
2) The number of vehicles required;
3) Sensitivity. To see whether the algorithms are sensitive with respect to the demand distribution, problem size, the increase of demand mean, and the increase of demand variance, etc.

3.3.2 Building the Common Ground for Comparison

In addition to the assumptions already made, we further assume that, when applying the dynamic programming to adopt optimum recourse policy, a partial delivery is allowed, and no penalty is imposed in case of a partial delivery.

3.3.2.1 Problem Generation

Problem size and distance matrix

We assume that the problem size ranges from 20-110, and the cost $c_{i,j}$ is the Euclidean distance between two points i and j. Each problem instance is randomly generated in the square $[0, 100]^2$, according to a uniform distribution. And we assume that the depot is situated at the center (50, 50).

Demand pattern

Based on the demand patterns used in the selected algorithms, we select both the discretized normal and the discrete uniform distribution for the following reasons:

1) Both of these two distributions are simple in the sense that the probability can be easily calculated for a given demand possibility.
2) To examine whether these algorithms are sensitive to different demand distributions.
3) When studying the algorithm performance with the increase of demand mean, the uniform distribution has a problem that with the increase of demand range, the variance also increases; while for normal distribution, we can examine the net effect of the demand mean.

In fact, the algorithm can be applied to any demand pattern, given that the probability associated with a certain demand distribution is known. For example, customer demands can be not identically distributed. However, for simplicity, we assume that all demands are identically distributed in this comparative study.

Number of instances

For each type of problems, the number of instances used in this study is 30. The performances of the heuristics are compared based on the same problem instances. The computational results: the expected route length, running time, number of vehicles used (in the multiple vehicle case), etc., which are the basis of comparison, are the averages of the results obtained from the 30 randomly generated problem instances.

Factors affecting the performance of algorithms

In this comparative study, we will examine the following situations:

1) Single vehicle routing and multiple vehicle routing problems.
2) Algorithm performance with the increase of problem size, with fixed demand pattern for both normal and uniform distributions.
3) Algorithm performance with the increase of demand mean, with fixed demand variance and problem size for normal distribution.
4) Algorithm performance with the increase of demand variance, with fixed demand mean and problem size for normal distribution.
5) Algorithm performance with the increase of demand mean and variance, with fixed problem size for uniform distribution.

3.3.2.2 Adaptations Made to the Original Algorithms

Bertsimas *et al.* (1995)'s cyclic heuristic

In Bertsimas *et al.* (1995), they did not specify which heuristic was used to construct the initial TSP solution. In this study, we employ the space-filling curve heuristic (Bartholdi and Platzman, 1982) to generate the initial TSP tour.

In their study, both 2-interchange and dynamic programming were included in their cyclic heuristic. However, computational results showed that, the improvement due to 2-interchange (less that 1%) is much smaller than that of the dynamic programming procedure (6%), though inclusion of either procedure increases running time by a factor of 2-3 times. They also proposed that, 2-interchange procedure requires that customer demands be identically distributed. For these reasons, we only use the dynamic programming to adopt the best recourse policy in this comparative study.

The original algorithm is designed for single vehicle routing. To make the comparison in the case of multiple-vehicle routing, we apply the dynamic programming partitioning procedure proposed by Yang *et al.* (2000) to partition a single route into multiple routes. After the partitioning, for each single route, the cyclic heuristic is used again to seek further improvements.

For Bertsimas *et al.* (1995), the algorithm can be executed in two ways: (1) By applying Equation (3.1) to calculate the expected cost, we first find the best sequence from cyclic heuristic, then apply dynamic programming recursion to adopt optimal recourse policy, this is called B-1 in Section 3.4; (2) We utilize the dynamic programming recursive procedure throughout the algorithm. The Equation (3.1) to calculate the expected cost (Bertsimas, 1995) is not used at all, this is called B-2 in Section 3.4.

Yang *et al.* (2000)'s algorithms

Similarly, in Yang *et al.* (2000), for the single vehicle routing and the route-first-cluster-second algorithms, they did not mention which specific insertion heuristic is used to form the initial TSP. We adopt the arbitrary insertion heuristic, because the following Or-opt procedure and possible other post improvement procedures will compensate for the relatively bad performance of the arbitrary insertion heuristic.

We also adapt cluster-first-route-second algorithm to the single vehicle routing algorithm by relaxing the constraint imposed on the maximum expected route length on each route. In Section 3.4, cluster-first-route-second algorithm is known as C1-R2, while the route-first-cluster-second one is called R1-C2.

Teodorovic and Pavkovic (1992)'s simulated annealing (SA) algorithm

For the SA algorithm in Teodorovic and Pavkovic (1992), the following adaptations must be made to make the comparison work:

1) Route failure can occur any times in each route rather than at most once.
2) The expression to calculate the expected cost in their original paper no longer works. To calculate the expected cost, we also adopt the dynamic programming recursion (Bertsimas *et al.*, 1995) to calculate the expected cost and the threshold values, and to determine the best recourse policy, because of its effectiveness in solving VRPSD.
3) For single vehicle routing, there are two ways to make the adaptation. One is that, the first stage SA is omitted, because its role in their original paper is to cluster the customers in different routes, this is known as SA-1 in Section 3.4. Secondly, we employ the first stage SA to find a satisfactory initial solution, then second stage SA is applied to seek further improvements, it is called SA-2 in Section 3.4.

3.4 Computational Results and Analysis

To get the computational results, the algorithms are coded in VISUAL C++, and run on a 500 MHZ Pentium II Processor with 128 MB RAM under Microsoft Windows 98.

Throughout this section, the following notations are used to represent the algorithms selected into the comparative study.

B-1: Bertsimas *et al*. (1995)'s algorithm, applying Equation (3.1) to calculate the expected cost, and dynamic programming recursion to adopt optimal recourse policy.

B-2: Bertsimas *et al*. (1995)'s algorithm, applying dynamic programming recursion to calculate both the expected cost and the optimal recourse policy.

C1-R2: Yang *et al*. (2000)'s cluster-first-route-second algorithm.

R1-C2: Yang *et al*. (2000)'s route-first-cluster-second algorithm.

SA-1: Teodorovic and Pavkovic (1992)'s simulated annealing algorithm with first stage SA being omitted.

SA-2: Teodorovic and Pavkovic (1992)'s simulated annealing algorithm with first stage SA being used to find a satisfactory initial solution.

3.4.1 Computational Results

3.4.1.1 Multiple Vehicle Routing Algorithms

We examine the demand pattern for both truncated normal distribution and uniform distribution. Originally, SA algorithm is included in the comparison. However, its performance is quite bad, with both expected cost and computational time several times greater than those of the other heuristics. Since it is obvious that the SA algorithm is the worst, it is included only in the comparison for single vehicle routing but not in the comparison for the multiple vehicle routing.

1) Uniform Distribution

Uniform Distribution --Performance with the increase of problem size

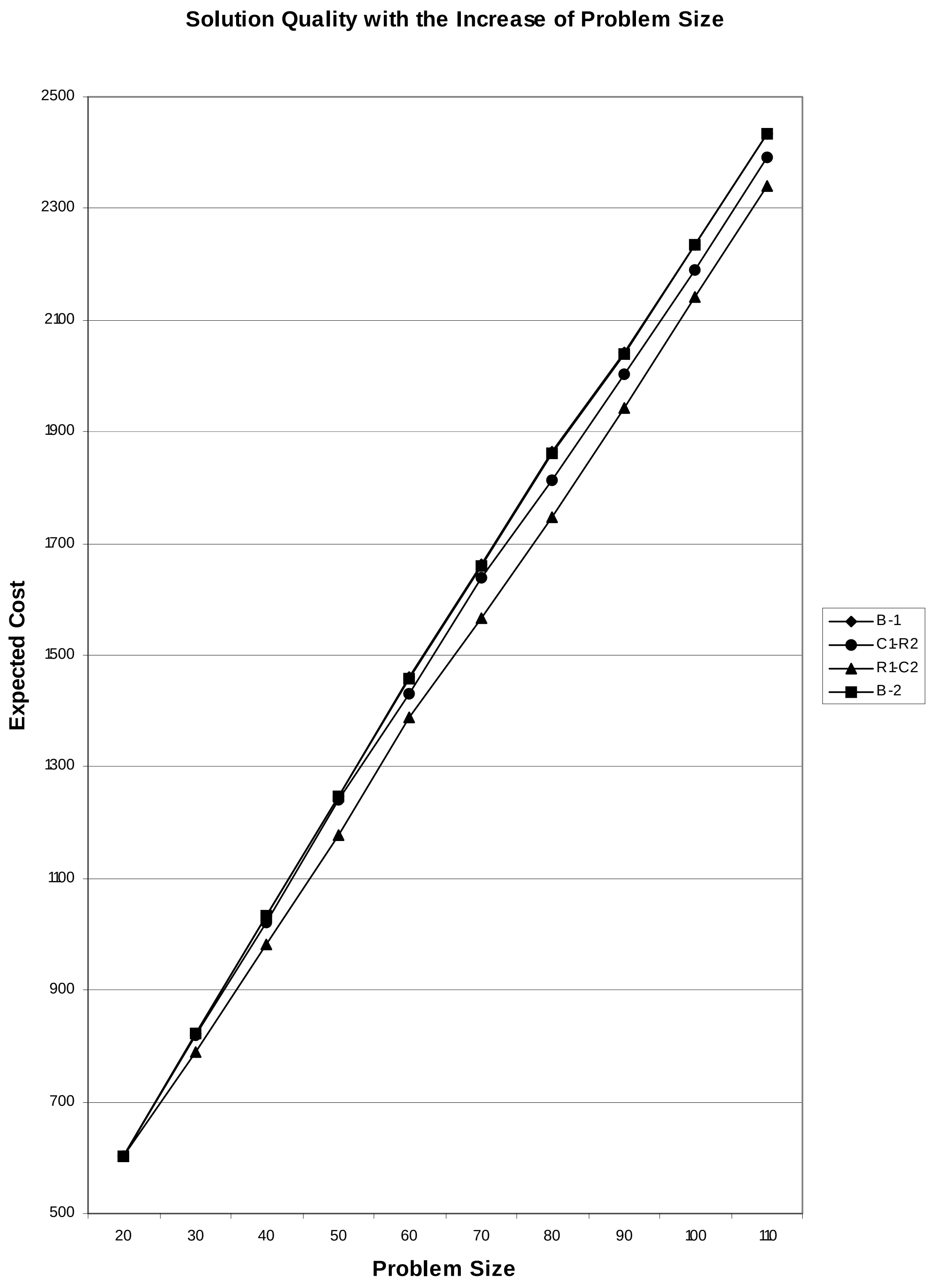


Figure 3.1 Expected Cost with the Increase of Problem Size
(Demands follow uniform distribution U[0, 20])

Here, we assume that all demands are generated from the uniform distribution. We examine two demand cases: U[0, 20] and U[0, 55], where *Case* 1 has relatively lower mean with respect to vehicle capacity *Q*. Figures 3.1 and 3.2 illustrate the solution quality and computational time of *Case*1 respectively. From Figure 3.1, it is clear that, with the increase of problem size, the expected cost increases linearly for all of the algorithms studied. The solution quality differs slightly when the problem size is small. When problem size becomes larger, R1-C2 performs slightly better than the other three algorithms. B-1 and B-2 perform almost equally the worst, leaving C1-R2 in between. Regarding the number of vehicles used, both R1-C2 and C1-R2 use fewer vehicles than B-1 and B-2.

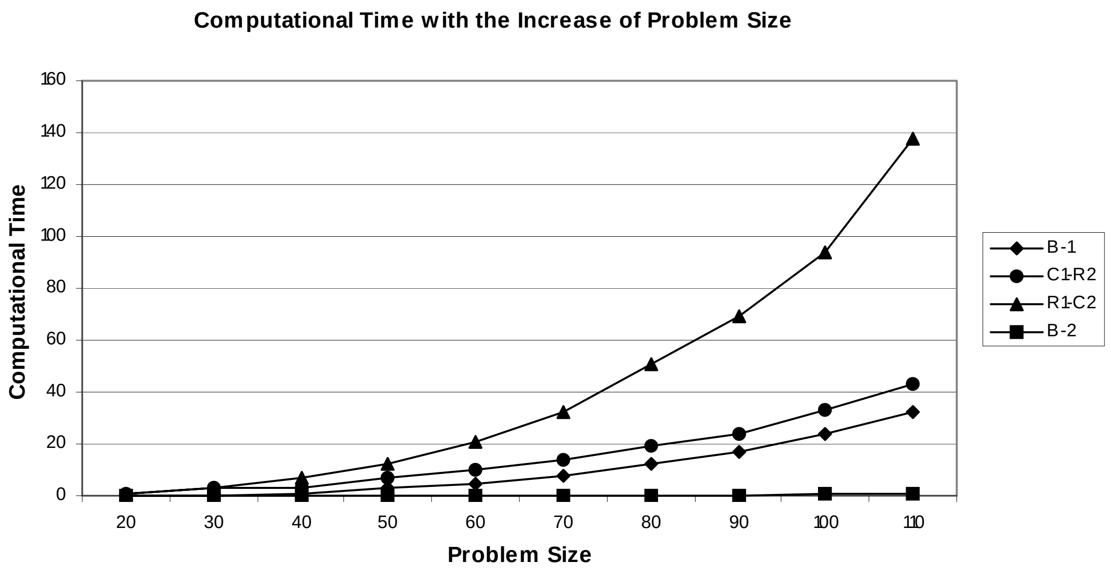


Figure 3.2 Computational Time with the Increase of Problem Size
(Demands follow uniform distribution U[0, 20])

Figure 3.2 shows the computational time taken by the algorithms against problem sizes. It is noted that B-2 takes very little time, while R1-C2 takes the longest time. For the other two algorithms, B-1 runs faster than C1-R2. Computational time of R1-C2 also increases fastest with the increase of problem size. Details of the average case performance regarding both solution quality and computational time can be seen in Table 3.1.

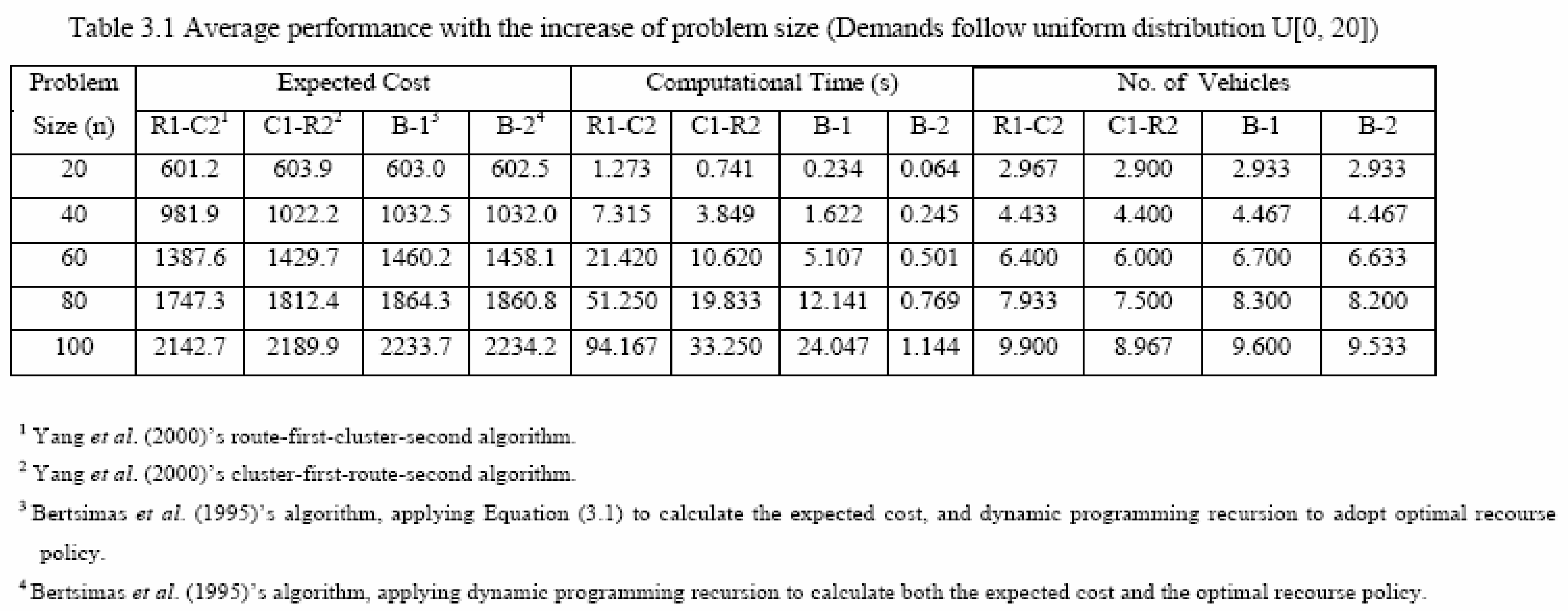


Table 3.1 Average performance with the increase of problem size (Demands follow uniform distribution U[0, 20])

Problem	Expected Cost				Computational Time (s)				No. of Vehicles			
Size (n)	R1-C2[1]	C1-R2[2]	B-1[3]	B-2[4]	R1-C2	C1-R2	B-1	B-2	R1-C2	C1-R2	B-1	B-2
20	601.2	603.9	603.0	602.5	1.273	0.741	0.234	0.064	2.967	2.900	2.933	2.933
40	981.9	1022.2	1032.5	1032.0	7.315	3.849	1.622	0.245	4.433	4.400	4.467	4.467
60	1387.6	1429.7	1460.2	1458.1	21.420	10.620	5.107	0.501	6.400	6.000	6.700	6.633
80	1747.3	1812.4	1864.3	1860.8	51.250	19.833	12.141	0.769	7.933	7.500	8.300	8.200
100	2142.7	2189.9	2233.7	2234.2	94.167	33.250	24.047	1.144	9.900	8.967	9.600	9.533

[1] Yang *et al.* (2000)'s route-first-cluster-second algorithm.

[2] Yang *et al.* (2000)'s cluster-first-route-second algorithm.

[3] Bertsimas *et al.* (1995)'s algorithm, applying Equation (3.1) to calculate the expected cost, and dynamic programming recursion to adopt optimal recourse policy.

[4] Bertsimas *et al.* (1995)'s algorithm, applying dynamic programming recursion to calculate both the expected cost and the optimal recourse policy.

In *Case* 2, both B-1 and B-2 seem to perform better than in *Case*1. The difference among the algorithms becomes even smaller in terms of expected cost and number of vehicles used. With respect to the computational time, R1-C2 also takes the most time, but B-1 ranks the second most in this case, followed by C1-R2, B-2 also takes the least time. This implies that, in *Case* 2 where demand may take higher value, solution quality of B-1 and B-2 become better, while B-1 also takes more computational time.

Uniform Distribution --Performance with the increase of mean and variance

Figure 3.3 illustrates how solution quality of the algorithms performs with the increase of demand mean and variance when the problem size is fixed at 60. Here the x-axis (MEAN/Q) is the ratio between the demand mean and the vehicle capacity, which represents the increase of demand mean and variance. Similar to the case with the increase of problem size, as the demand mean and variance increase, the expected cost also increases linearly for all the algorithms examined. The difference between the algorithms is marginal. Moreover, as the mean increases, the difference between B-1 (the worst) and the rest of the algorithms becomes even smaller. Nevertheless, we still can see that R1-C2 performs the best among all the algorithms studied.

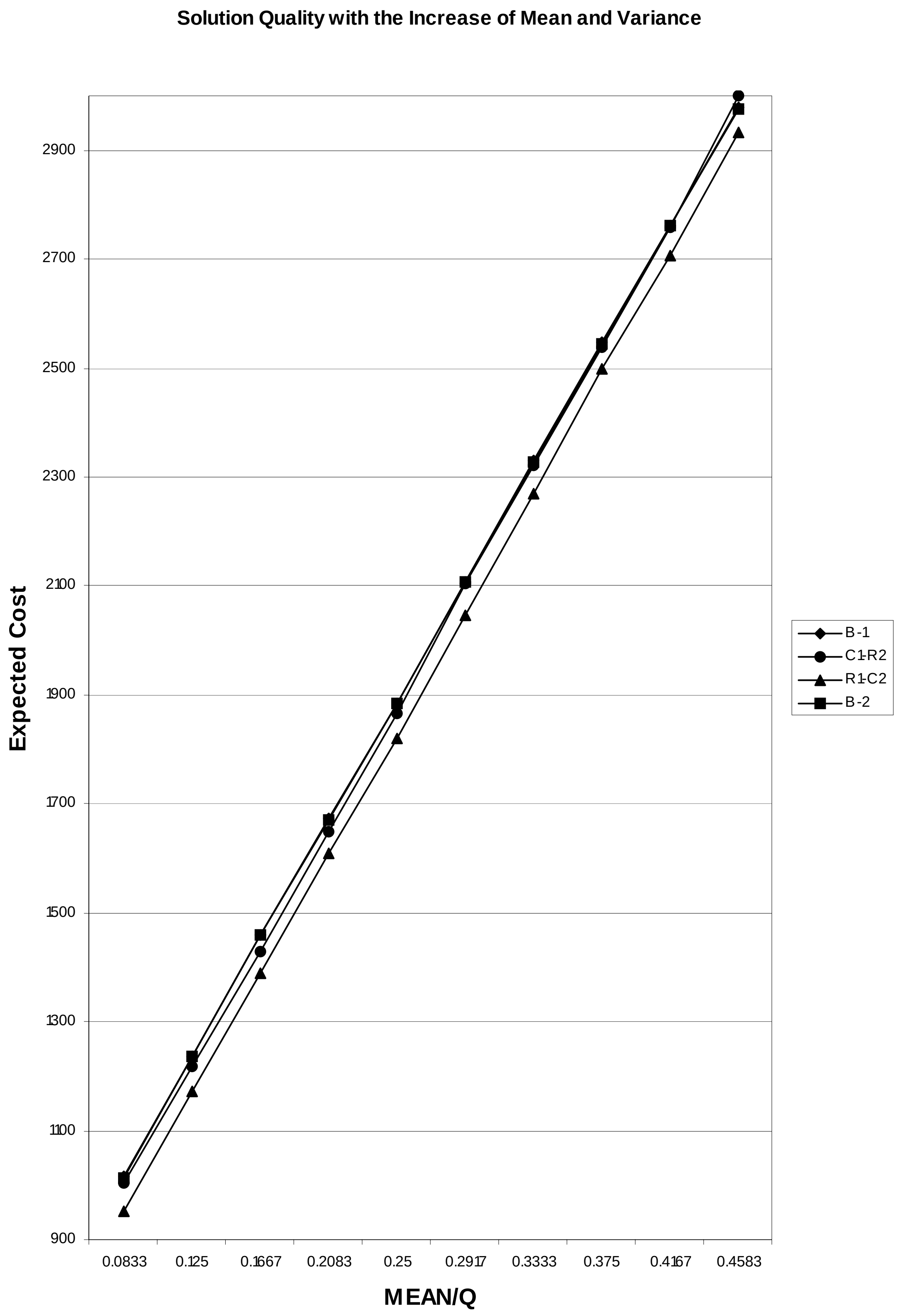


Figure 3.3 Expected Cost with the Increase of Demand Mean and Variance (Problem size *n* = 60)

Computational time taken is similar to that illustrated in Figure 3.2, as can be observed in Figure 3.4. R1-C2 consumes the most time, and B-2 takes the least time. As demand mean grows, the computational time of B-1 increases fastest, while C1-R2 increases very slowly. Therefore, when demand mean approaches 25% or more of the vehicle capacity, B-1 takes more time than C1-R2. Detailed numerical results are presented in Table 3.2.

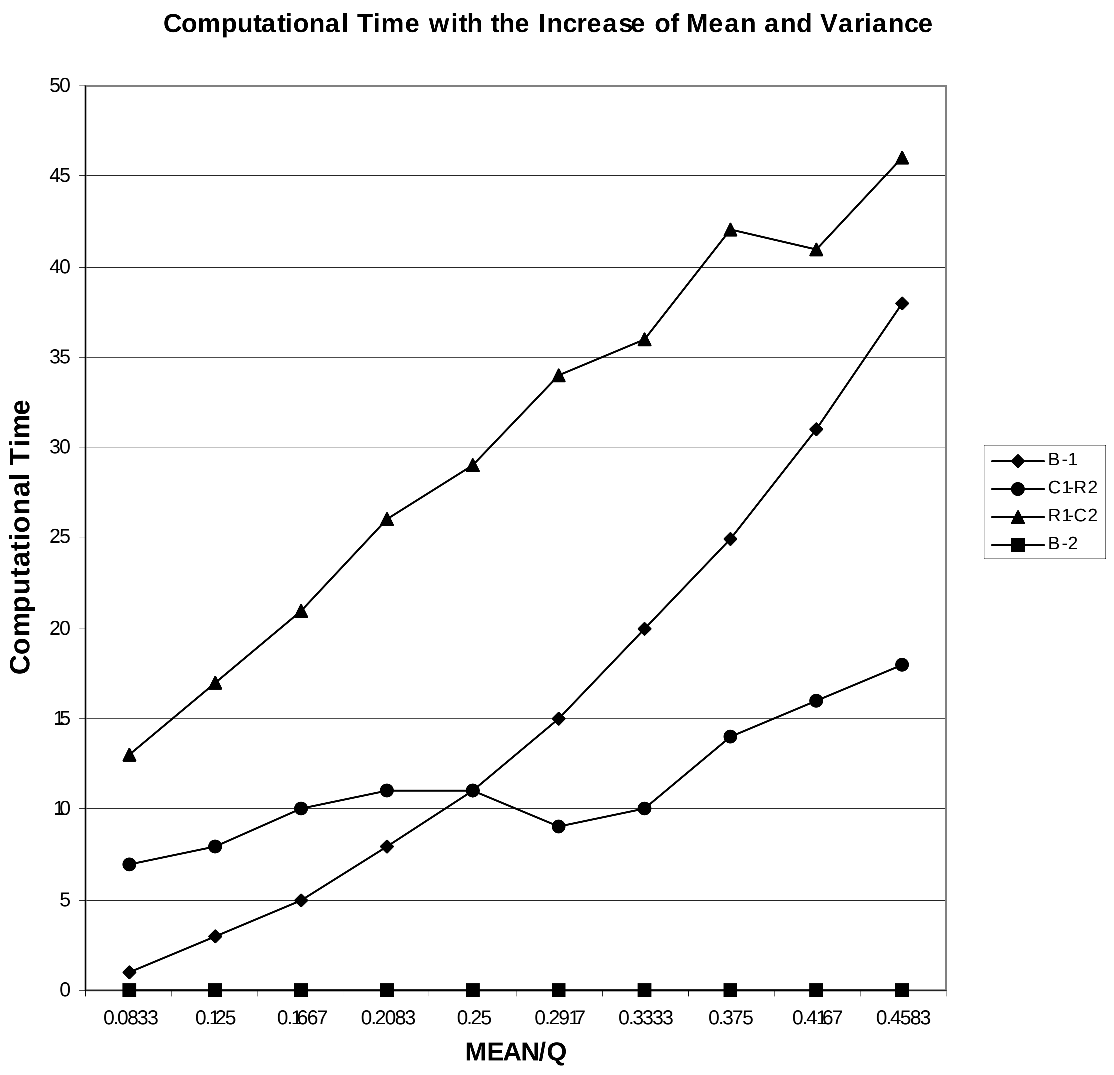


Figure 3.4 Computational Time with the Increase of Demand Mean and Variance (Problem size *n* = 60)

Table 3.2 Average performance with the increase of demand mean and variance (Problem size $n = 60$)

Mean	Expected Cost				Computational Time (s)				No. of Vehicles			
/Q*	R1-C2[1]	C1-R2[2]	B-1[3]	B-2[4]	R1-C2	C1-R2	B-1	B-2	R1-C2	C1-R2	B-1	B-2
0.0833	950.8	1004.4	1016.0	1014.4	13.157	7.124	1.023	0.421	4.767	4.833	4.600	4.600
0.1667	1387.7	1429.7	1460.2	1458.1	21.420	10.264	5.145	0.556	6.400	6.000	6.700	6.633
0.2500	1819.3	1866.8	1885.7	1883.3	29.267	11.433	11.362	0.584	8.1667	7.567	8.067	8.133
0.3333	2268.4	2321.8	2329.7	2328.4	36.432	10.890	20.521	0.657	10.033	9.167	10.033	9.967
0.4167	2707.7	2758.9	2762.7	2761.4	41.548	16.211	31.342	0.738	11.533	10.833	11.533	11.500

* The ratio between the demand mean and the vehicle capacity, which represents the increase of demand mean and variance.

[1] Yang *et al.* (2000)'s route-first-cluster-second algorithm.

[2] Yang *et al.* (2000)'s cluster-first-route-second algorithm.

[3] Bertsimas *et al.* (1995)'s algorithm, applying Equation (3.1) to calculate the expected cost, and dynamic programming recursion to adopt optimal recourse policy.

[4] Bertsimas *et al.* (1995)'s algorithm, applying dynamic programming recursion to calculate both the expected cost and the optimal recourse policy.

2) Normal distribution

The performance of the algorithms when demands follow normal distribution is similar to that when demands follow uniform distribution. Some observations are as follows.

Normal Distribution -- Performance with the increase of problem size

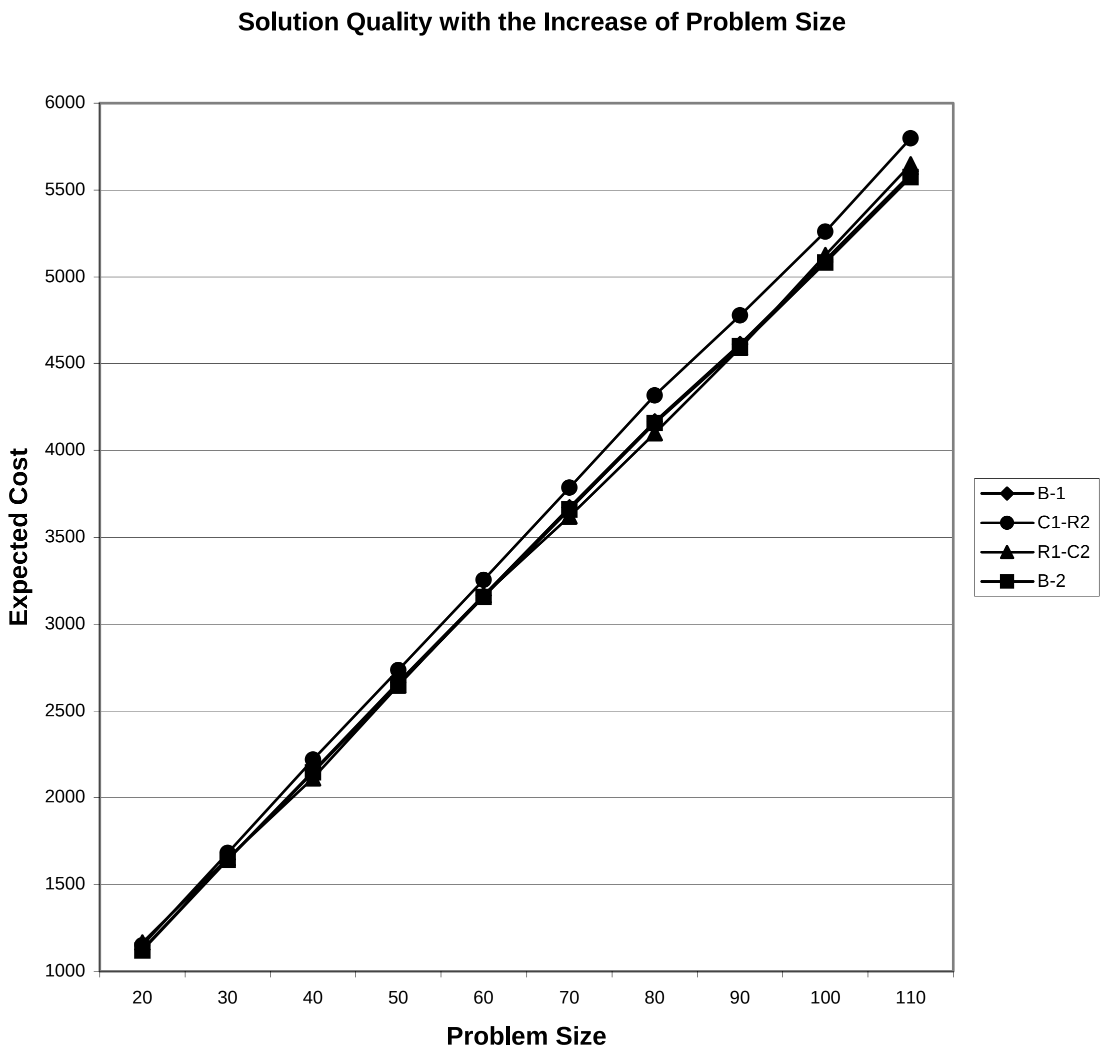


Figure 3.5 Expected Cost with the Increase of Problem Size
(Demands follow normal distribution N(30,25))

Here, we assume that all demands are generated from the truncated normal distribution N(30, 25). In terms of solution quality, in this case, as problem size becomes larger, the algorithms R1-C2, B-1 and B-2 tend to outperform the C1-R2 algorithm. C1-R2 performs the worst, with the performance of the other algorithms being very similar to one another. Detailed results for the solution quality are illustrated in Figure 3.5.

With respect to the computational time, in this case, B-1 takes the most time as compared to the other three heuristics; B-1 also increases fastest as problem size grows. R1-C2 ranks the second most, B-2 still takes the least time. The difference among the number of vehicles used is marginal in this case.

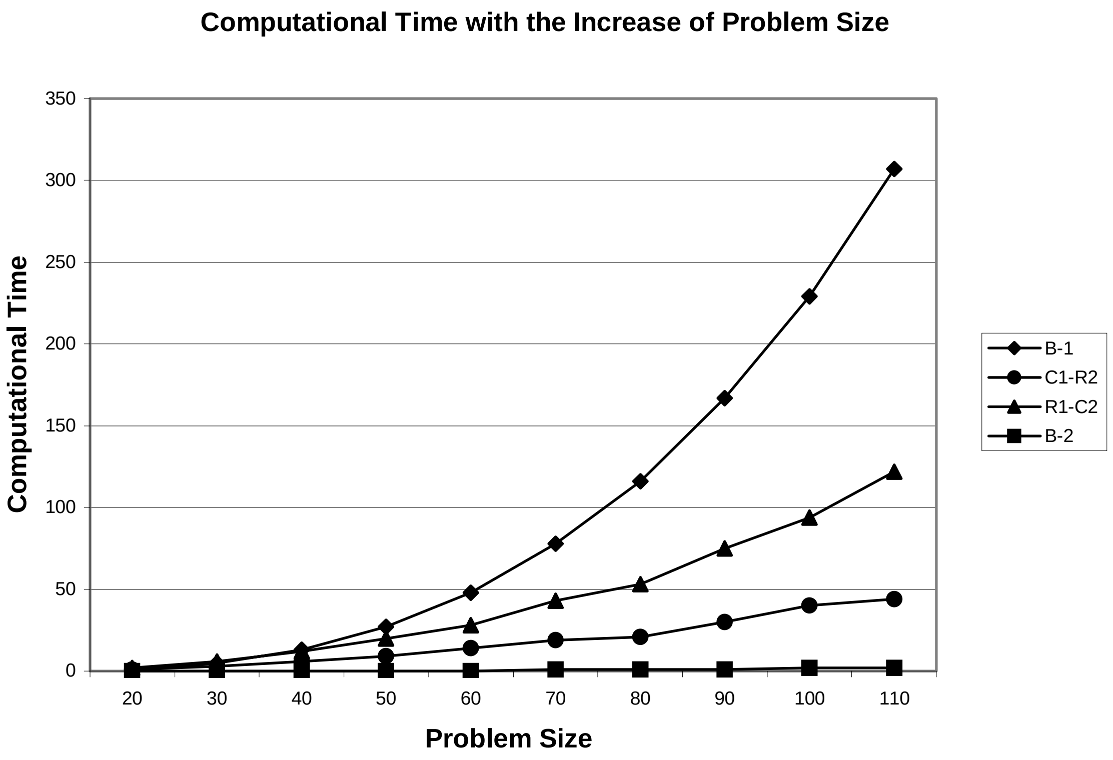


Figure 3.6 Computational Time with the Increase of Problem Size
(Demands follow normal distribution N(30,25))

Normal Distribution -- Performance with the increase of mean

We examined the case where problem size is 20. Here the x-axis (MEAN/Q) is the ratio between the demand mean and the vehicle capacity, which represents the increase of demand mean. Results for solution quality and computational time are illustrated in Figure 3.7 and Figure 3.8 respectively. For most of the demand mean ranges, the expected cost increases linearly with the increase of demand mean. However, when the demand mean approaches 75% or more of the vehicle capacity, the expected cost begins to decrease. Result also reveals that, as demand mean grows larger, both B-1 and B-2 performs better than R1-C2 and C1-R2. The performance of the later two algorithms is quite similar in this case.

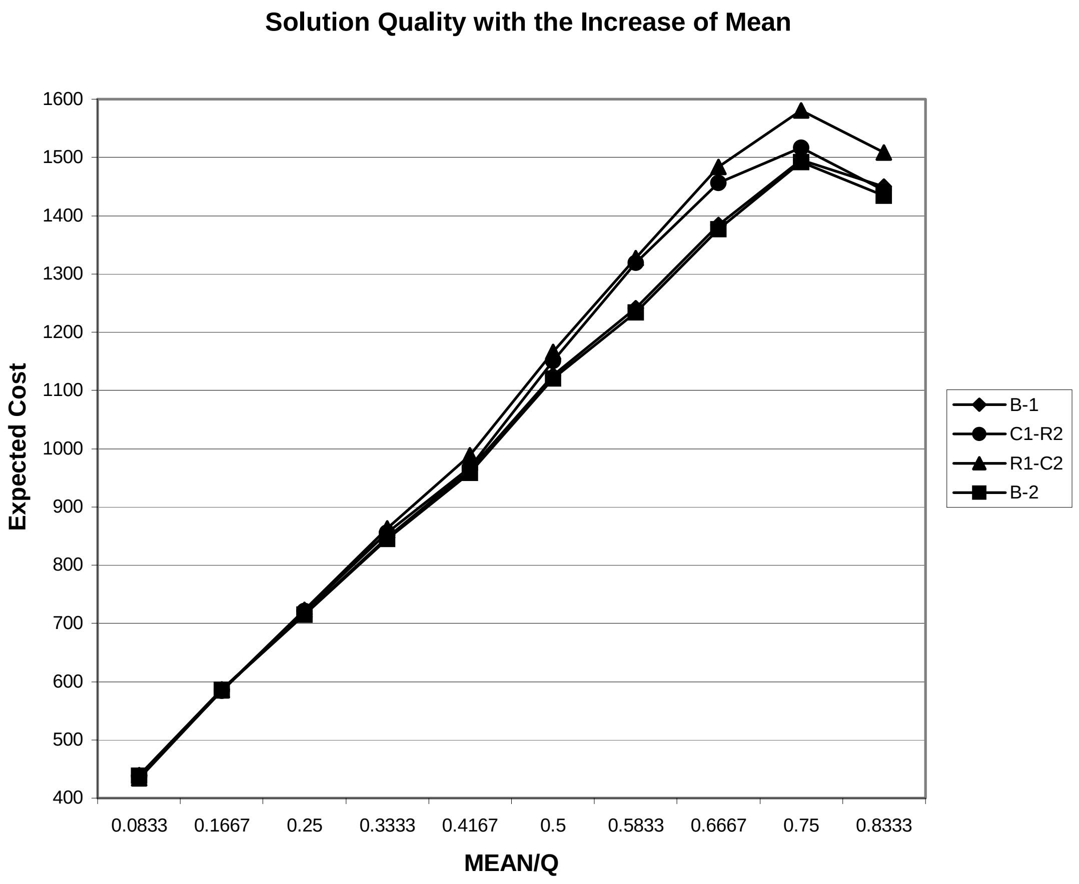


Figure 3.7 Expected Cost with the Increase of Demand Mean
(Problem size n = 20, demand variance = 25)

Since the problem size is relatively small, the computational time is less than 3 CPU seconds for all the four algorithms. Based on findings obtained when demands follow uniform distribution, we can expect that, when problem size is large, the difference in terms of both solution quality and computational time among algorithms can grow larger.

Normal Distribution -Performance with the increase of variance

Our results show that, with the increase of demand variance, the expected cost, the number of vehicles used and the computational time are all almost the same, which indicates that variance does not have much influence on the algorithm performance. Figure 3.8 shows how the algorithms behave with the increase of demand variance with respect to solution quality.

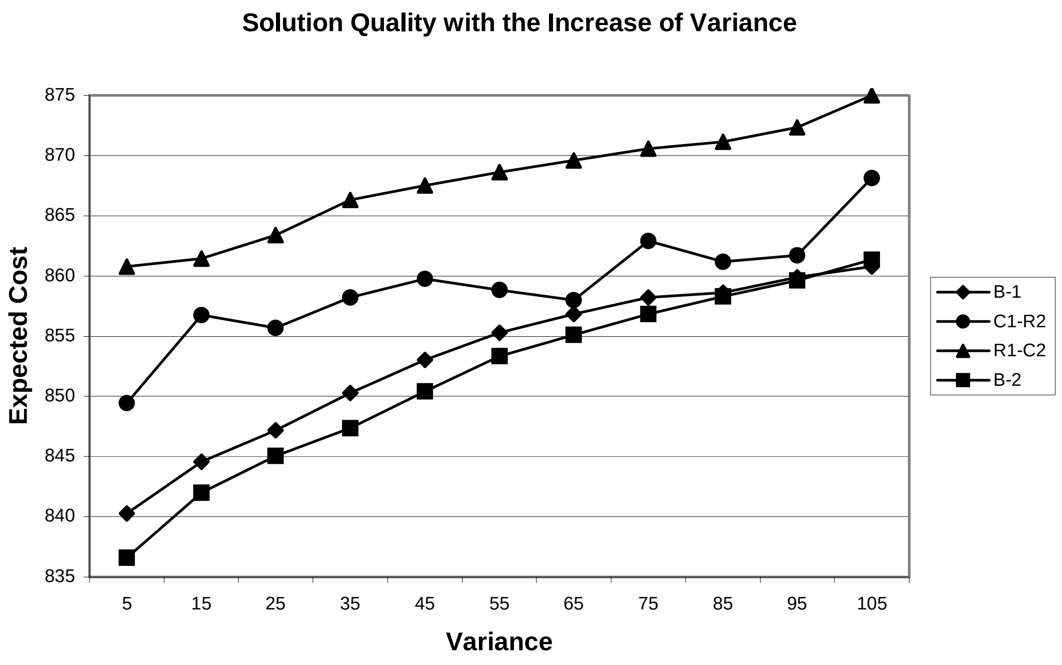


Figure 3.8 Expected Cost with the Increase of Demand Variance
(Problem size *n* = 20, demand mean = 20)

3.4.1.2 Single Vehicle Routing Algorithms

From the comparison of the multiple vehicle routing, it is obvious that the algorithms are not quite sensitive to the demand distribution type. Therefore, for single vehicle routing problem, we only examine the demand pattern following uniform distribution.

Performance with the increase of problem size

Similar to the multiple vehicle routing, the expected cost increases linearly as problem size grows larger, as can be seen in Figure 3.9.

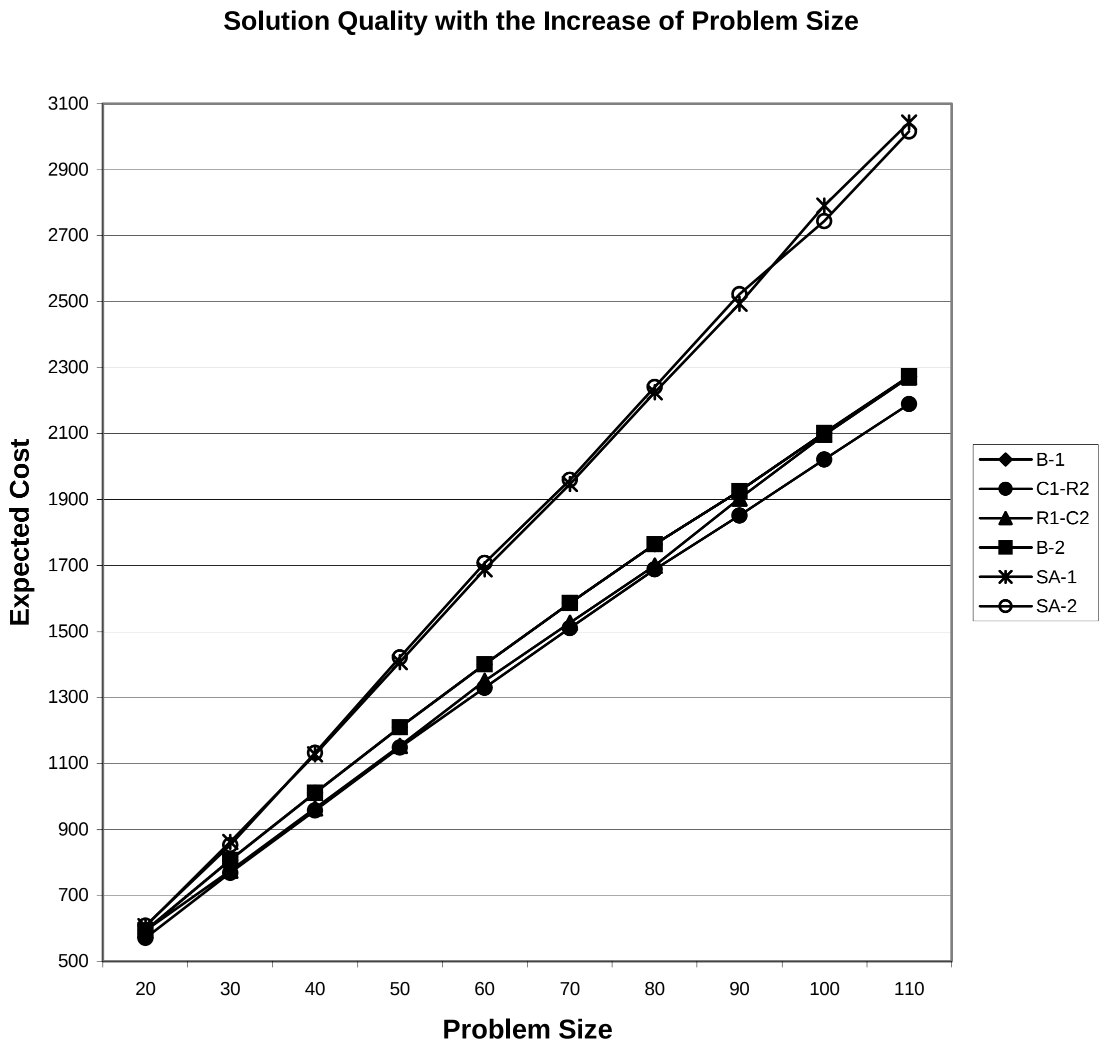


Figure 3.9 Expected Cost with the Increase of Problem Size
(Single vehicle, demands follow uniform distribution U[0,20])

However, in this case, solution quality of C1-R2 seems to be the best, and SA-1 and SA-2 appear to be the worst. When the problem size is small, the difference between the algorithms is little; while as problem size grows, both SA-1 and SA-2 tend to divert greatly from the others.

Regarding the computational time, both implementations of SA take a lot of computational time. Though the computational time increases with the increase of problem size, it fluctuates greatly. The solution quality of SA-1 and SA-2 is very close, but SA-2 seems to take even more time. As we can expect from the solution quality, C1-R2 takes relatively more time compared to the other three algorithms, B-2 takes the least time, B-1 ranks the second least, and computational time of R1-C2 is very similar to that of C1-R2. Refer to Figure 3.10. Detailed computational results can be found in Table 3.3.

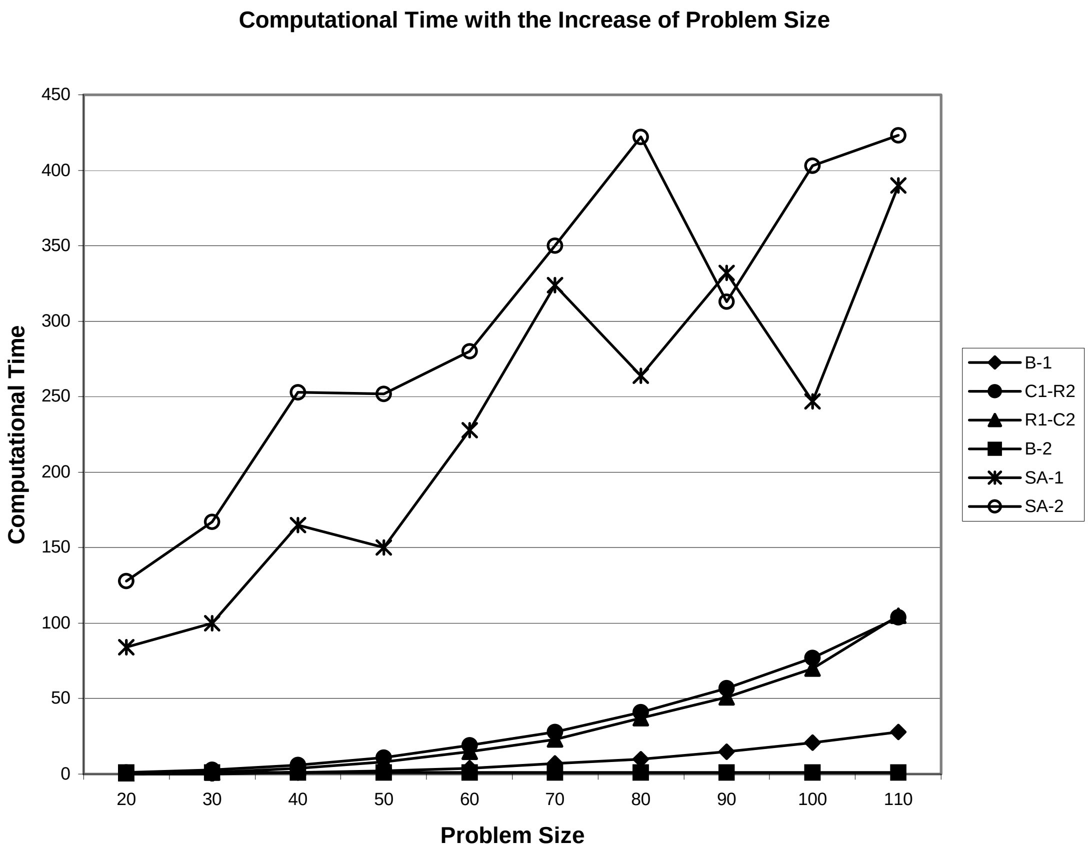
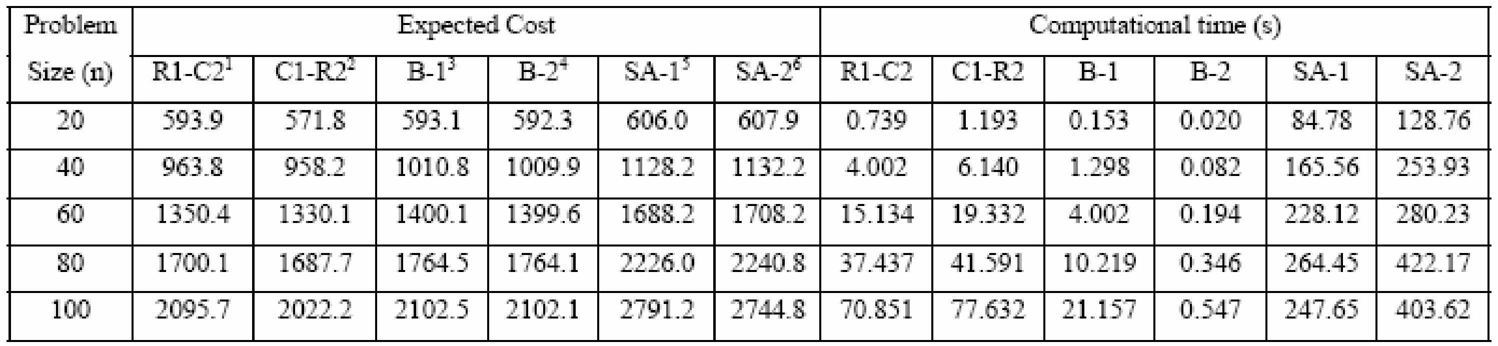


Figure 3.10 Computational Time with the Increase of Problem Size
(Single vehicle, demands follow uniform distribution U[0,20])

Table 3.3 Average performance with the increase of problem size (Demands follow uniform distribution U[0, 20], single vehicle)



Problem	Expected Cost						Computational time (s)					
Size (n)	R1-C2[1]	C1-R2[2]	B-1[3]	B-2[4]	SA-1[5]	SA-2[6]	R1-C2	C1-R2	B-1	B-2	SA-1	SA-2
20	593.9	571.8	593.1	592.3	606.0	607.9	0.739	1.193	0.153	0.020	84.78	128.76
40	963.8	958.2	1010.8	1009.9	1128.2	1132.2	4.002	6.140	1.298	0.082	165.56	253.93
60	1350.4	1330.1	1400.1	1399.6	1688.2	1708.2	15.134	19.332	4.002	0.194	228.12	280.23
80	1700.1	1687.7	1764.5	1764.1	2226.0	2240.8	37.437	41.591	10.219	0.346	264.45	422.17
100	2095.7	2022.2	2102.5	2102.1	2791.2	2744.8	70.851	77.632	21.157	0.547	247.65	403.62

[1] Yang *et al*. (2000)'s route-first-cluster-second algorithm.

[2] Yang *et al*. (2000)'s cluster-first-route-second algorithm.

[3] Bertsimas *et al*. (1995)'s algorithm, applying Equation (3.1) to calculate the expected cost, and dynamic programming recursion to adopt optimal recourse policy.

[4] Bertsimas *et al*. (1995)'s algorithm, applying dynamic programming recursion to calculate both the expected cost and the optimal recourse policy.

[5] Teodorovic and Pavkovic (1992)'s simulated annealing algorithm with first stage SA being omitted.

[6] Teodorovic and Pavkovic (1992)'s simulated annealing algorithm with first stage SA being used to find a satisfactory initial solution.

Performance with the increase of mean and variance

As demand mean and variance grow larger, the expected cost of the algorithms grows approximately linearly, see Figure 3.11. Here the x-axis (MEAN/Q) is the ratio between the demand mean and the vehicle capacity, which represents the increase of demand mean and variance.

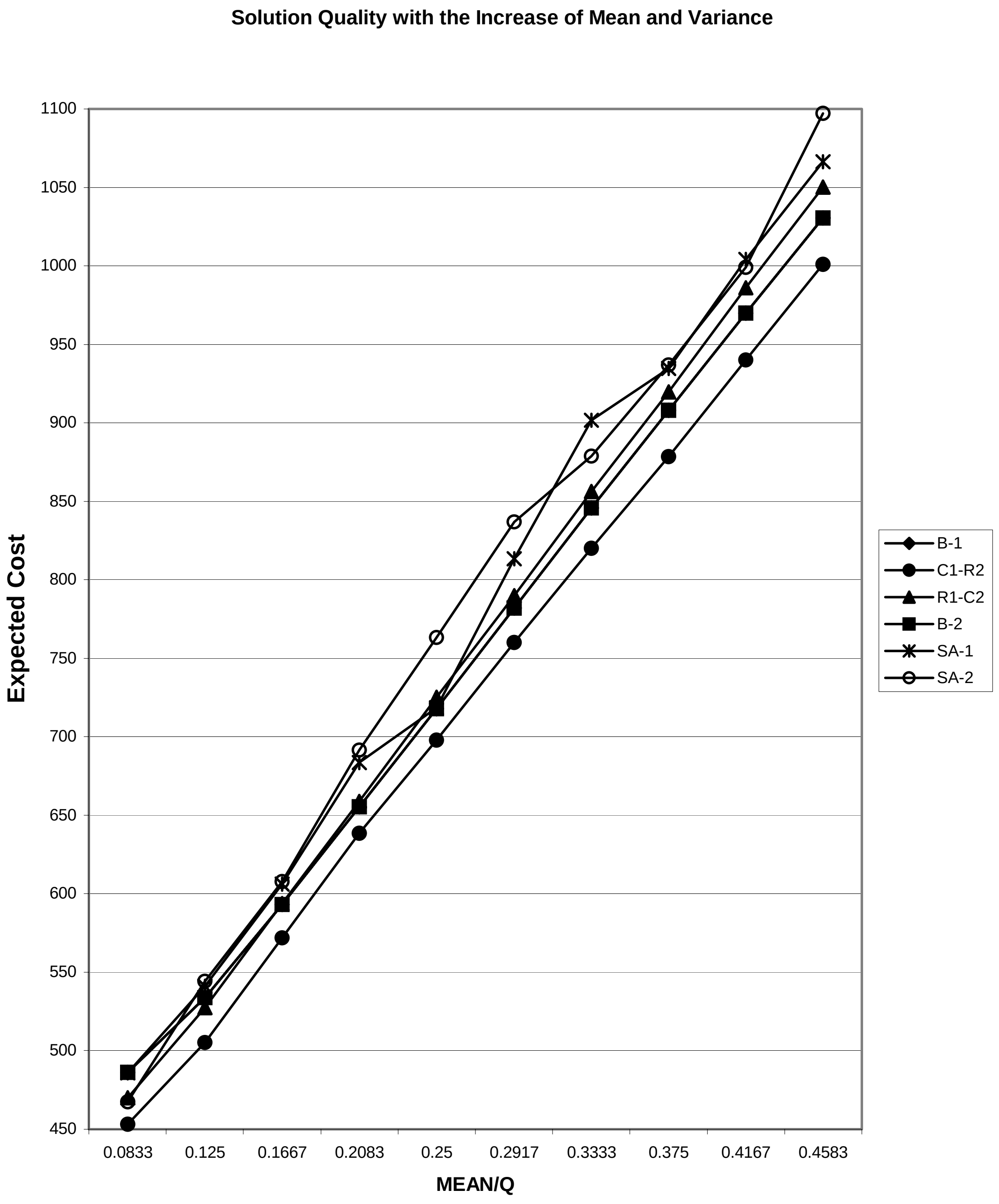


Figure 3.11 Expected Cost with the Increase of Demand Mean and Variance (Single vehicle, problem size *n* =20)

However, in this case, the expected cost of both SA-1 and SA-2 has some fluctuations. Results in Table 3.4 show that, C1-R2 is best, SA-1 and SA-2 are almost equally the worst, leaving R1-C2, B-1 and B-2 in between. The difference between the algorithms is not substantial. However, we can predict that, as the problem size increases, the difference between SA and the other algorithms can grow larger.

Since the problem size is small, the computational time is very little except that of the SA-1 and SA-2. The computational time for SA-1 and SA-2 is not only very long but also fluctuates greatly, without clear trend of increase with the increase of demand mean and variance.

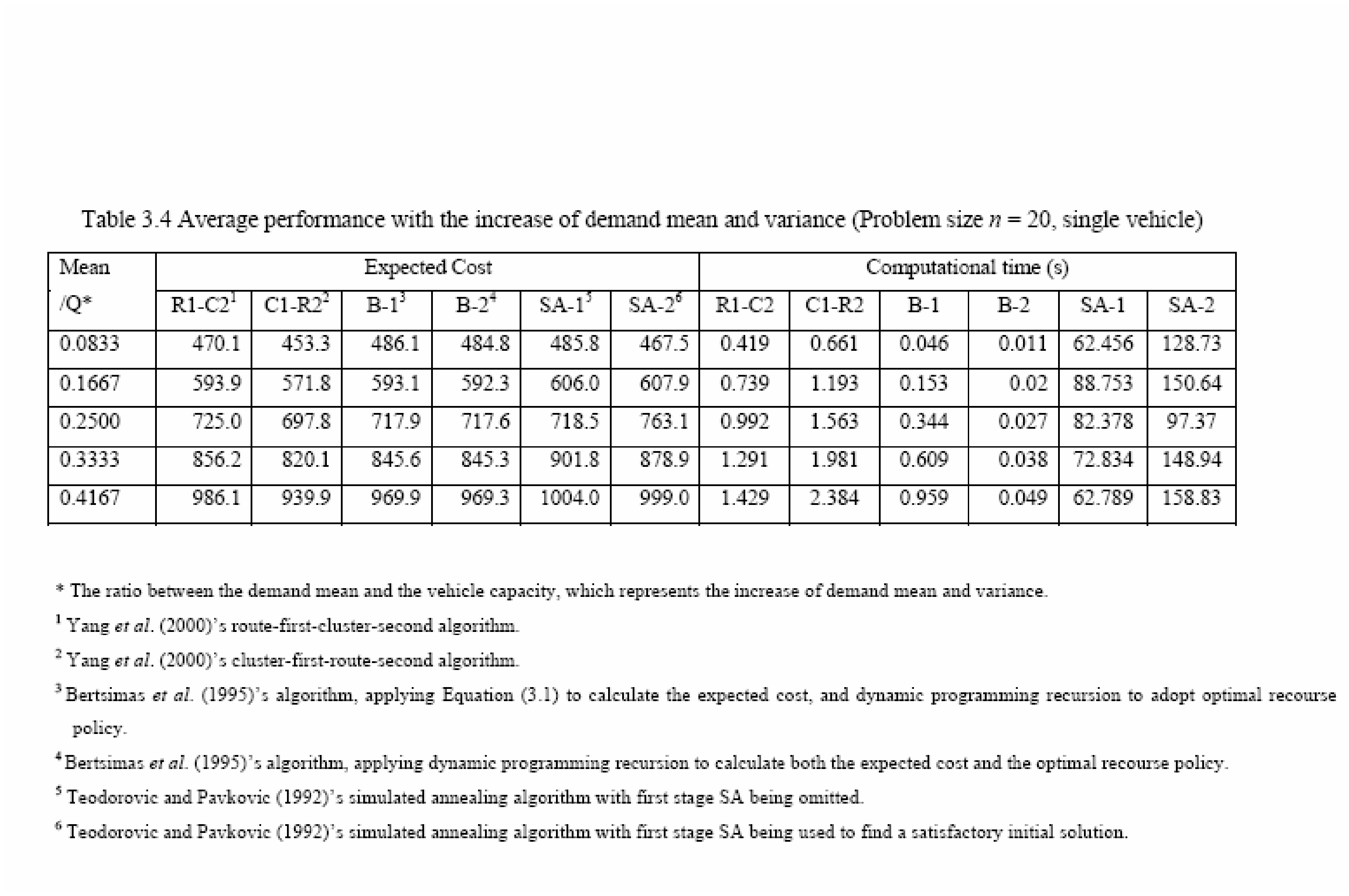


Table 3.4 Average performance with the increase of demand mean and variance (Problem size $n = 20$, single vehicle)

Mean	Expected Cost						Computational time (s)					
/Q*	R1-C2[1]	C1-R2[2]	B-1[3]	B-2[4]	SA-1[5]	SA-2[6]	R1-C2	C1-R2	B-1	B-2	SA-1	SA-2
0.0833	470.1	453.3	486.1	484.8	485.8	467.5	0.419	0.661	0.046	0.011	62.456	128.73
0.1667	593.9	571.8	593.1	592.3	606.0	607.9	0.739	1.193	0.153	0.02	88.753	150.64
0.2500	725.0	697.8	717.9	717.6	718.5	763.1	0.992	1.563	0.344	0.027	82.378	97.37
0.3333	856.2	820.1	845.6	845.3	901.8	878.9	1.291	1.981	0.609	0.038	72.834	148.94
0.4167	986.1	939.9	969.9	969.3	1004.0	999.0	1.429	2.384	0.959	0.049	62.789	158.83

* The ratio between the demand mean and the vehicle capacity, which represents the increase of demand mean and variance.

[1] Yang *et al.* (2000)'s route-first-cluster-second algorithm.

[2] Yang *et al.* (2000)'s cluster-first-route-second algorithm.

[3] Bertsimas *et al.* (1995)'s algorithm, applying Equation (3.1) to calculate the expected cost, and dynamic programming recursion to adopt optimal recourse policy.

[4] Bertsimas *et al.* (1995)'s algorithm, applying dynamic programming recursion to calculate both the expected cost and the optimal recourse policy.

[5] Teodorovic and Pavkovic (1992)'s simulated annealing algorithm with first stage SA being omitted.

[6] Teodorovic and Pavkovic (1992)'s simulated annealing algorithm with first stage SA being used to find a satisfactory initial solution.

3.4.2 Performance Analysis of the Algorithms

1) Common observations for all of the algorithms

Sensitivity analysis

An algorithm is said to be sensitive if it performs well in one case, but badly in the others. Generally, the algorithms examined are not very sensitive to demand distribution, problem size, increase of demand mean and/or variance. Some observations are summarized below.

Demand distribution. For all the four algorithms, under the same problem size, the expected cost when demand follows normal distribution is slightly lower than that demand follows uniform distribution. This observation is consistent with the result of Bertsimas *et al.* (1995). Moreover, B-1 seems to take more computational time in the case of normal distribution. On the contrary, Yang *et al.* (2000)'s algorithms seem to take more time in the case of uniform distribution.

Increase of demand mean. The solution quality of B-1 and B-2 tends to improve more compared with the other algorithms as demand mean increases. However, as demand mean grows, the computational time of B-1 also increases fastest.

Increase of problem size. In multiple vehicle routing, R1-C2 performs better when problem size is large with respect to other algorithms. In single vehicle routing, C1-R2 performs better when there are more demand points.

Increase of demand variance. Results indicate that the increase of demand variance does not have much influence on the performance of the algorithms.

Solution quality

Problem size. The expected cost increases linearly as problem size grows.

Demand mean. As demand mean increases, the expected cost also increases, with the exception that, when demand follows normal distribution and when demand mean is 75% or more of the vehicle capacity, the expected cost begins to decrease. This may be due to the following reason: When demand follows normal distribution and demand mean is relatively large compared with the vehicle capacity, the probability that demand takes very small value is almost zero; while when demand follows uniform distribution, the probability that demand takes any value from zero to maximum demand possibility is the same.

Demand variance. The solution quality is not influenced by the demand variance.

We observe that in single vehicle routing, with respect to solution quality, C1-R2 performs best, SA-1 and SA-2 perform the worst, leaving R1-C2, B-1 and B-2 almost equally in between. While in multiple vehicle routing, R1-C2 seems superior to the others. B-1 and B-2 behave better when demand mean is high.

Computational time

Except for B-2 in multiple vehicle routing and SA-1 and SA-2 in single vehicle routing, it is usually the case that, the algorithm with better solution quality takes relatively more time.

Comparing B-1 with B-2, the solution quality is almost the same. However, B-1 is much more time consuming. As described in Section 3.1, Equation (3.1) is used to calculate the expected cost. At each demand point, as stated in Bertsimas (1992), the two probabilities γ_i and δ_i are calculated recursively, which is very time consuming.

When comparing B-2 with Yang *et al.* (2000)'s algorithms, though all of them employ dynamic programming procedure to evaluate the expected cost, B-2 takes very little time, because the cyclic heuristic only involves comparing the *n* permutations, while in Yang *et al.* (2000)'s algorithms, the Or-opt and the other possible improving procedures, require many comparisons. Therefore, though Yang *et al.* (2000)'s algorithms adopted the approximate dynamic programming procedure to calculate the insertion cost, it still takes more time.

Similarly, in single vehicle routing, the solution quality of SA-1 does not differ greatly from SA-2, though SA-2 takes more computational time. This means that, the first stage SA which is used to find a satisfactory initial solution does not seem to be quite effective.

2) Characteristics of certain algorithms

Our results for Yang *et al.* (2000)'s algorithms show that, for the multiple vehicle routing, R1-C2 is slightly better than C1-R2 with respect to solution quality. However, the former takes more computational time than the later. This result agrees with the result in Yang *et al.* (2000).

The result from SA is not good in comparison with the other heuristic algorithms with respect to both solution quality and computational time. Moreover, it seems that there are some randomness involved in both the solution quality and computational time. The poor performance of this SA may be due to the following reasons:

1) In the first stage of SA, generating route sets randomly may not be a good idea comparing with other possible more systematic neighborhood generation mechanisms.

2) In the second stage, the neighborhood structure (just exchange one node with another randomly selected node) also may not be so efficient.

The randomness in generating the initial tour and the neighborhood solutions is also the main reason of the long computational time. Trying other neighborhood generation mechanism, such as Or-opt, may be worthwhile.

3.5 Summary and Conclusions

In this chapter, we present a comparative study on some algorithms for solving the vehicle routing problems with stochastic demands. We examine the performance of these algorithms in various situations with respect to problem size and demand pattern, in order to determine which algorithm is superior to the others. Our study reveals that, except for the SA algorithm, the differences among the other algorithms examined are not substantial. From the aspect of practical application, our study suggests that B-2 is the best candidate, because it takes very little computational time, with solution quality comparable to the other heuristics. However, if the decision maker regards the solution quality as more important, in single vehicle routing, C1-R2 is a good choice; while in multiple vehicle routing, R1-C2 tends to be better. Due to the good solution quality of the Yang *et al.* (2000)'s algorithms, they are employed as a performance measurer in Chapter 4 to evaluate the metaheuristics proposed for the VRPSD.

Chapter 4

Metaheuristics for Vehicle Routing Problem with Stochastic Demands

In Chapter 3, a comparative study was carried out for the algorithms presented in Bertsimas *et al.* (1995), Yang *et al.* (2000) and Teodorovic and Pavkovic (1992). The comparative study shown that the SA algorithm based on the original neighborhood generation method and cooling schedule performed quite badly in comparison with the other heuristic algorithms with respect to both solution quality and computational time. However, this does not mean that SA algorithm is not suitable for VRPSD, because the performance of a SA algorithm depends greatly on the neighborhood generation mechanism and the cooling schedule. Thus, in this chapter, we try to examine how modern metaheuristics behave for the VRPSD.

To examine the performance of the proposed metaheuristics, the computational results of the proposed algorithms are compared with the results from the algorithms presented in Yang *et al.* (2000). In Yang *et al.* (2000), under the assumption that demand follows discrete triangular distribution, they claimed that, for single vehicle case, with small problem size (n = 8 and n = 10), their algorithm provides solutions with at most 1% average deviation from the optimal solution; for multiple vehicle case, with problem size at 10, 12 and 15, the deviation from the optimum is at most 3.3%. Moreover, computational results of the comparative study in Chapter 3 also shown that, among the heuristics compared, Yang *et al.* (2000)'s algorithms perform the best in terms of solution quality. Since Yang *et al.* (2000)'s algorithms perform very well for the VRPSD, we employ them as a performance measurer in this chapter.

4.1 Metaheuristics for Single VRPSD

4.1.1 Initial Solution and Generation of Neighborhood Solutions

One important element that determines the performance of metaheuristics is the neighborhood generation mechanism. In this study, we employ the same neighborhood generation method for each of the three algorithms so that we can examine the performance of the algorithm itself. The neighborhood generation method is described below.

Given a current solution, a neighborhood solution can be obtained by relocating a node from its current location to a position between two other neighbor nodes in the tour. Bentley (1992) called this move a 2.5-opt, because it is a special and simple form of 3-opt (Lin, 1965). Here, in this study, we restrict the relocation of the node to only one of its nearest neighbors rather than elsewhere in the tour. Specifically, for each node, find ϖ nearest neighbors and build a neighbor list of size ϖ. To generate a neighborhood solution for node *i*, first generate a random number *b* in the range 1 to ϖ, then find the node *j* which is b^{th} nearest neighbor in node *i*'s neighbor list, remove node *i* from its current position and put it before or after node *j*. The neighbor list size ϖ should be set to a suitable value. If ϖ is too small, the solution will have difficulty in escaping from the local optimum; if ϖ is too large, the procedure tends to become a random search. In this study, the neighbor list size ϖ is set as follows:

$$\varpi = \begin{cases} 4 & \text{if } n \le 20 \\ \lceil n/10 \rceil & \text{otherwise} \end{cases}$$

To examine whether the metaheuristics are sensitive to the initial solution, we generate the initial solution in the following three ways: a random tour, a tour generated from the nearest neighbor heuristic (Rosenkrantz *et al.*, 1977), and a tour generated based on the space-filling curve heuristic (Bartholdi and Platzman, 1982). The effect of the initial solution on the algorithm is discussed in Section 4.3.1.

4.1.2 The Simulated Annealing and Threshold Accepting Algorithms

The name simulated annealing (SA) was originated from the physical annealing process. Kirkpatrick *et al.*(1983) proposed that the concept form the basis of an optimization technique for combinatorial and other problems. The basic idea of SA is to provide for small random perturbations and to compute the amount of changes in objective function value. A typical feature of a SA algorithm is that, in addition to accepting improvements in objective function value, it also accepts deteriorations in objective function value with a certain probability. Initially, a large value of a control parameter, called temperature, is used to evaluate the probability of accepting an unfavorable move. At the start, large deteriorations can be accepted; as temperature decreases, only smaller deteriorations will be accepted. This feature makes SA, in contrast to the local search algorithms, be able to escape from local minima without losing the favorable features of local search algorithms: simplicity and flexibility. SA is one of the metaheuristics proposed for the combinatorial problems. Ever since its introduction, SA has been applied to a large number of combinatorial optimization problems. Some of the applications of SA in solving VRP include: Alfa *et al.* (1991), Osman (1993), Hiquebran *et al.* (1994), etc.

Threshold accepting (TA), introduced in Dueck and Scheuer (1990), is a deterministic analog to SA. The main difference is the rule of accepting worse neighborhood solutions. SA accepts a worse solution according to the acceptance probability; while TA accepts a worse one if the deterioration is within a certain threshold value. The threshold value is decreased as the algorithm proceeds, and finally approaches zero when the algorithm terminates. One advantage of TA over SA is that, it is not necessary to compute probabilities or to make random decisions, making the algorithm even simpler. In Dueck and Scheuer (1990), they claim that TA yields better results than SA. An extensive introduction to TA is given in Winker (2000).

4.1.2.1 Thermal Equilibrium Condition in the SA and TA Procedures

To ensure the asymptotic convergence to the global optimal solution, the thermal equilibrium of the SA (TA) procedure should be attained at each temperature. One way to achieve thermal equilibrium is that, at a certain temperature, the objective function value does not change for several consecutive transitions. In Teodorovic and Pavkovic (1992), the thermal equilibrium is defined as follows: within each temperature, they set a maximum number of generations to perform; within each generation, a fixed number of transitions are carried out. Once a generation is finished, the total cost of all the transitions within the generation is recorded and another generation is performed. If the relative percentage change in the total cost between the current and the previous generations is less than a predefined thermal equilibrium factor ε , the thermal equilibrium is reached at the particular temperature and the temperature is lowered. In Hiquebran *et al.* (1994), thermal equilibrium is defined in a similar way. However, they used the cost of the final transition of each generation as an evaluation measure. In our study, we employ the one used in Teodorovic and Pavkovic (1992).

4.1.2.2 The Procedure Involved in the SA and TA Algorithms

The SA algorithm involves the following steps:

Step 1: Initialization:

1.1: Generate an initial tour. Calculate the expected cost according to the DP recourse procedure described in Section 3.1.3. Denote it as L_{old}.

1.2: Select an initial temperature $t_0 > 0$ and a temperature reduction factor α. Set $t_i = t_0$, $i = 0$, and set total number of different temperatures needed to *no_t*.

Step 2: Set initial *generation_count* = 0, total number of generations = *no_gen*; set the length of the total routes in the last generation $Total_{last} = \infty$, and the length of the total routes in the current generation $Total_{current} = 0$.

Step 3: Select a node in the current route; generate a neighborhood solution as described in Section 4.1.1. Calculate the expected cost for the new route, denoted as L_{new}. Calculate the difference between the new and the old route, $\Delta = L_{new}\text{-}L_{old}$. If $\Delta < 0$, go to Step 5.

Step 4: Generate a uniformly distributed random number $rand \in [0,1]$. Calculate the acceptance probability $prob = exp(-\Delta/t_i)$. If $rand < prob$, go to Step 5; otherwise, keep the old route and its expected cost. Go to Step 6.

Step 5: Record the new route and its expected cost.

Step 6: Update the total route length of the current generation $Total_{current}$ by adding the length of the current route to $Total_{current.}$ If the neighborhood solutions for all n nodes have been examined 3 times, go to Step 7**.** Otherwise go to Step 3 to examine the next node**.**

Step 7: Check whether thermal equilibrium has been reached at temperature t_i by testing $\left|Total_{current} - Total_{last}\right| / Total_{current} < \varepsilon$, where ε is a predefined thermal equilibrium factor. If the above inequality holds, go to Step 8. Otherwise, update the total route length of the last generation by replacing $Total_{last}$ with $Total_{current}$ if $Total_{current} < Total_{last}$. Increase the *generation_count* by 1. If *generation_count* > *no_gen* go to Step 8 else go to Step 3**.**

Step 8: If the expected cost of the current solution does not change over the last 10 consecutive temperatures t_i, or a predefined maximum number of temperatures *no_t* have been performed, stop the algorithm and return the current route and its expected cost as the final solution.

Step 9: Lower the temperature by setting $t_{i+1} = t_i * \alpha$. Set $i = i+1$ and go to Step 2.

The steps involved in the TA algorithm only differ from those in SA in the following ways:

1) In Step 1.2, instead of setting a temperature, we set a threshold value *threshold*.
2) In Step 4, we just compare Δ with the threshold value, if $\Delta <$ *threshold*, go to Step 5; otherwise, keep the old route set and its expected cost and then go to Step 6.
3) The threshold value decreases at the same reduction rate α. That is, *threshold* = *threshold*$*\alpha$.

4.1.2.3 Parameter Setting in the SA Algorithm

Initial temperature t_0

To ensure that the final solution is independent of the starting solution, the initial temperature should be high enough; however, a too high initial temperature will undoubtedly increase the computational time. An appropriate t_0 can be determined by ensuring the initial acceptance ratio $A_0 = \psi(t_0)$, which is defined by the ratio between the number of accepted transitions and the number of all transitions, is close to 1. First start at a sufficiently high t_0, and lower it rather quickly until the acceptance ratio reaches a prescribed value. Our computational experience shows that, $t_0 = 40$ is an appropriate initial temperature for the SA algorithm.

Temperature reduction factor α

For the temperature reduction factor α, literature suggests to use values between 0.8 and 0.99. We set $\alpha = 0.9$.

Final temperature t_f and number of temperatures *no_t*

The final temperature should be low enough to guarantee that no worse solutions are accepted. Suppose we consider t_f to be low enough, when cost increase as large as 0.5 is accepted with very low probability (such as *prob* = 0.001). The final temperature can be calculated based on the equation $prob = exp(-\Delta/t_f)$, which is about 0.072. With the initial temperature $t_0 = 40$ and $\alpha = 0.9$, the total number of temperatures *no_t* can be determined by the equation $t_f = t_0 * \alpha^{no-t}$, which leads to *no_t* = 60.

Thermal equilibrium factor ε and number of generations at each temperature *no_gen*

The number of generations at each temperature is usually related to the size of the neighborhoods or the solution space. In this study, we set *no_gen* = 300. However, we perform $3*n$ transitions within each generation. Therefore, with the increase of problem size, the number of transitions performed at each temperature also increases. Nevertheless, *no_gen* = 300 is actually the maximum number of generations set for each temperature, we allow the actual number of generations performed to vary from temperature to temperature, because not every segment of the cooling process is equally important. In the cooling process, most useful work is done in the middle of the schedule. Thus, it may be advantageous to search the beginning and the end parts of the temperature range quickly, using the time saved to allow slower and more elaborate cooling in the middle. In this study, this is done by properly setting the parameter ε, which determines whether thermal equilibrium is reached. By repeatedly fine-tuning, we set $\varepsilon = 0.2$ for the first and the last 10 temperatures, and set $\varepsilon = 0.03$ for the 40 temperatures in between. As a result, less number of generations is performed at the beginning and the end of the cooling procedure. This is reasonable, as we can see from Table 4.1, during the first 10 temperatures, temperature drops from 40 to about 13.9, and the maximum allowable increase of the objective function value drops from 280 to 97.6. If we set ε to be a very low value, it is very difficult to reach the thermal equilibrium in this situation, which will result in long computational time. For the last 10 temperatures, both temperature value and the maximum allowable increase in cost are quite low, which indicates that the algorithm has already started to converge toward local optimum, therefore we should speed up the cooling process and quickly converge the solution to the targeted local optimum. Since different problems may not require the same number of temperatures to converge, to further save the computational time, we terminate the algorithm when the solution quality does not improve for about 10 consecutive temperatures.

Table 4.1 Temperature & maximum allowable increase in cost in different cooling stages

Number of Temperatures	1	10	50	60
Temperature value	40	13.95	0.21	0.07
Maximum Allowable Increase in Cost	280	97.63	1.44	0.50

4.1.2.4 Parameter Setting in the TA Algorithm

For TA, the parameters are set similarly except for the threshold value. In TA, threshold value corresponds to temperature in SA. The initial threshold value is set equal to 40 when $n < 60$ and 60 otherwise. We also set the threshold reduction rate α = 0.9.

4.1.3 The Tabu Search Algorithm

Tabu search is an iterative procedure designed for the solution of optimization problems. It was first proposed by Glover and has been used to solve a wide range of hard combinatorial optimization problems (Glover and Laguna, 1997). To avoid being trapped in local optima, one needs to accept a move even when it makes the current solution worse. However, when a worse solution is accepted, cycling may occur; that is, the move may take us right back to the local optimum we just left or to some other recently visited solutions. This is where the TS algorithm comes in by keeping information about the moves most recently made in one or more tabu lists, to forbid or penalize certain moves that would return to a recently visited solution. Three general components are usually incorporated in TS to direct the search trajectory: tabu restrictions and aspiration criteria, short-term and long-term memory structures, and intensification and diversification strategies. Tabu restrictions are the tabu lists that keep track of recently examined solutions to avoid cycling. Aspiration criteria provide exceptions to the general tabu rules to allow a move to be performed provided the move is considered as good enough. Intensification is the exploration of a promising neighborhood, such as the neighborhood of a previously found good solution; while diversification aims to search in previously unexplored regions of the solution space. Both of these strategies are realized by means of short-term and long-term memory structures. TS is one of the major metaheuristics that have been successfully applied in solving combinatorial optimization problems. Successful applications of TS in the area of vehicle routing problems can be found in Taillard (1993), Osman (1993), Fiechter (1994), Gendreau *et al.* (1994), Potvin *et al.* (1996), Duhamel *et al.* (1997), Taillard *et al.* (1997), Gendreau *et al.* (1999), etc.

4.1.3.1 The Main Components and the Parameter Setting in the TS Algorithm

To get the best performance in terms of both solution quality and computational time, we repeatedly fine-tune the parameters in the algorithm. The total number of iterations, *NoIter*, is set to 600. We set the other parameters as follows:

The neighborhood size *neighbor_size*

For a given current solution X, the complete neighborhoods of X, $N(\mathrm{X})$, contain all combinations of repositioning each of the n nodes according to the neighborhood generation method. At the current iteration, when we try to find a solution Y among a selected subset of $N(\mathrm{X})$ for possible acceptance, we should consider the tradeoff between the quality of Y and the effort spent on finding it, which is mainly determined by the size of the neighborhood. In this study, we restrict the number of neighborhood solutions generated from the current solution to *neighbor_size*. That is, we first randomly select *neighbor_size* nodes, then generate a neighborhood solution for each selected node. Our computational experience shows that best result in terms of both solution quality and computational time can be obtained when we set *neighbor_size* as follows:

$$neighbor\_size = \begin{cases} \lceil n/2 \rceil & \text{if } n \le 20 \\ \lceil n/3 \rceil & \text{otherwise} \end{cases}$$

Tabu list - recency based memory

Suppose in the current solution, we accept a move in which a node is removed from its current position and put to one of its nearest neighbors. The move that tries to reposition the same node again is tabu for the next *NoTabu* iterations. Specifically, we can use *TabuList*$_i$ to record the tabu status for each node *i*. In this study, the tabu tenure *NoTabu* is set as follows:

$$NoTabu = \begin{cases} 5 & \text{if } n \le 20 \\ \lceil n^{1/2} \rceil & \text{otherwise} \end{cases}$$

This means that as problem size increases, tabu tenure also increases to avoid searching certain solutions too often and therefore search all possible regions of the solution space.

Diversification - Frequency based memory

The strategy of diversification encourages the search process to examine unvisited regions and to generate solutions that are significantly different from those recently visited. We use *FreqDiver*$_{ij}$ to record the frequency of repositioning node *i* from its current location to the location immediately before or after its j^{th} nearest neighbor. Once such a move is accepted, the frequency *FreqDiver*$_{ij}$ is incremented by 1. Then we penalize such a move by adding a diversification penalty, *Diver_Penalty* = α_{Diver} * *FreqDiver*$_{ij}$, to the expected cost of this move, where α_{Diver} is a predetermined diversification factor. Certainly, we should not avoid visiting the previously visited moves forever. Therefore, after every *NoReset* (set to 100 in this study) iterations, we reset the frequency variable $FreqDiver_{ij}$ to 0. And all moves are eligible again in terms of frequency based memory.

In this study, the diversification factor α_{Diver} is set dependent on the performance of the move. If the move is better than the current solution, we set α_{Diver} = 0.5; otherwise, we set α_{Diver} = 2. In this way the worse solutions are more heavily penalized and hence they are less likely to occur frequently.

Intensification

Intensification aims to direct the search process to examine the neighbors of an elite solution. One intensification strategy employed in this study is to keep a current best solution. When the current best solution has not been improved for *NoNonImp* iterations, the current best solution is employed as the current solution, because its immediate neighborhoods are attractive regions, therefore should be searched more thoroughly.

In this study, we set *NoNonImp* = 200 iterations.

Aspiration criteria

In the searching process, we keep record of a current best solution. If a move is found to be better than the current best solution, it is accepted without considering its tabu status.

4.1.3.2 Procedure Involved in the TS Algorithm

The TS algorithm involves the following steps:

Step 1: Initialization

1.1 Generate an initial tour. Calculate the expected cost according to the DP recourse procedure described in Section 3.1.3.

1.2 Set *iteration_count* = 0; initialize *TabuList*$_i$, *FreqDiver*$_{ij}$.

Step 2: Set *neighbor_count* = 0, $z_{BestMove} = \infty$.

Step 3: Randomly select a node in the current route, generate a candidate move and calculate its expected cost *z*. If the move satisfies the aspiration criteria go to Step 5; if it is tabu go to Step 6; otherwise, go to Step 4.

Step 4: Calculate *Diver_Penalty* for this neighborhood move. Set $z = z +$ *Diver_Penalty*. If $z \geq z_{BestMove}$ go to Step 6.

Step 5: Mark the current candidate move as the best move, and set $z_{BestMove} = z$.

Step 6: Increment the *neighbor_count* by 1. If *neighbor_count* < *neighbor_size* go to Step 3.

Step 7: If a best move has been selected, update the *TabuList*, *FreqDiver* and the current solution. If the current best solution is worse than the selected best move, update it. If the current best solution has not been improved for *NoNonImp* iterations, employ the current best solution as the current solution.

Step 8: Increment the *iteration_count* by 1. If *iteration_count* < *NoIter* go to Step 2**.**

Step 9: Stop the algorithm and return the current best solution as the final solution.

4.2 Simulated Annealing and Threshold Accepting Algorithms for Multiple VRPSD

4.2.1 Generation of Neighborhood Solutions

Similar to the neighborhood structure described in Section 4.1.1 for single vehicle case, here, we also adopt the 2.5-opt (Bentley, 1992) move to generate neighborhood solutions. The difference is that, we allow the relocation within the node's own tour and other tours as well: a node *i* can move within its own route or to other existing routes, depending on which route node *j*, the selected neighbor of node *i*, is in. This neighborhood generation method allows the elimination of an existing route.

As in the single vehicle case, the neighbor list size ϖ changes with the problem size *n*. Moreover, it also changes with different cooling stages of the SA (TA) procedure. At the beginning of the search procedure, transitions with large deteriorations in cost should be provided for possible acceptance. As the search proceeds, the ability that SA (TA) can accept large deteriorations in cost also decreases. Therefore, the neighbor list size ϖ should also decrease. In our problem, we divide the SA (TA) procedure into 3 stages; the corresponding neighbor list size ϖ is as follows:

$$\varpi = \begin{cases} \lceil n/5 \rceil & \text{first stage} \\ \lceil n/10 \rceil & \text{second stage} \\ \lceil n/20 \rceil & \text{third stage} \end{cases}$$

4.2.2 Determining the Number of Vehicles and the Initial Solution

In this study, the number of vehicles *m* is a decision variable. However, when constructing the initial route set for the SA and TA, we have to roughly determine the number of vehicles in advance. The neighborhood generation method, as described in Section 4.2.1, allows the elimination of an existing route. Therefore, we can set an upper bound $\bar{m}$ for the number of vehicles *m* required by applying results from the comparative study in Chapter 3. The initial solution of SA (TA) can be obtained by first constructing a TSP tour for all nodes by using a TSP construction heuristic (nearest neighbor heuristic, for example), then partitioning the TSP tour into $\bar{m}$ tours.

4.2.3 Dealing with the Route Length Constraint

To satisfy the constraint that the total expected route length does not exceed a prescribed value *L*, the search should be guided in favor of feasible solutions. This is realized by adding a penalty element to the objective function. Suppose the expected route length of a certain route π is $L(\pi)$, the penalized objective function would then be $F(\pi) = L(\pi) + \rho * [L(\pi) - L]^{+}$, where $[x]^{+}$ = *max* (0, *x*), and ρ is a positive parameter.

With respect to the route length constraint, in order to diversify the searching process, it is not harmful to allow some infeasible solutions at the beginning. However, to ensure that the final solution is feasible, parameter ρ should be set appropriately. In this study, ρ is set to a relatively small value at the beginning and increased linearly with the increase of iterations. Nevertheless, infeasible solutions still may occur in this case though with very low probability. To overcome this problem, we add another threshold accepting procedure (TA-infeasibility) along with the SA (TA) procedure. In other words, we evaluate a solution according to two criteria: the solution quality and the feasibility. This TA-infeasibility procedure allows infeasible solutions provided that the total amount exceeded is within the threshold value. As threshold value decreases, TA-infeasibility procedure can control accepting infeasible solutions in terms of the extent of the infeasibility.

4.2.4 The Procedure Involved in the SA and TA Algorithms

The SA algorithm involves the following steps:

Step 1: Generate an initial route set by the procedure stated in Section 4.2.2. Calculate the expected route length using the DP recourse procedure described in Section 3.1.3. Denote it as L_{old}.

Step 2: Select an initial temperature $t_0 > 0$, a temperature reduction parameter α and an initial threshold value *threshold_infeasible*0 for the TA-infeasibility procedure. Set $t_i = t_0$, $i = 0$, and total number of different temperatures needed to *no_t*.

Step 3: Set initial *generation_count* = 0, total number of generations = *no_gen*; set the length of the total routes in the last generation $Total_{last} = \infty$, and the length of the total routes in the current generation $Total_{current} = 0$.

Step 4: Select a node in the current route set; generate a neighborhood solution as described in Section 4.2.1. For each route in the current route set, examine whether the expected route length exceeds the predefined limit. Calculate the total amount exceeded *total_exceed* for the whole route set. If *total_exceed* > *threshold_infeasible*, keep the old route set and its expected route length. Go to step 7; otherwise, calculate the expected route length for the new route set, L_{new}. Calculate the difference between the new and the old route set, $\Delta = L_{new}-L_{old}$. If $\Delta < 0$, go to Step 6.

Step 5: Generate a uniformly distributed random number $rand \in [0,1]$. Calculate the acceptance probability $prob = exp(-\Delta/t_i)$. If $rand < prob$, go to Step 6; otherwise, keep the old route set and its expected route length. Go to step 7.

Step 6: Record the new route set and its expected route length.

Step 7: Update the total route length of the current generation $Total_{current}$ by adding the length of the current route set to $Total_{current}$. If the neighborhood solutions for all *n* nodes have been examined 5 times, go to Step 8. Otherwise go to Step 4 to examine the next node**.**

Step 8: Check whether thermal equilibrium has been reached at temperature t_i by checking whether $\left|Total_{current} - Total_{last}\right| / Total_{current} < \varepsilon$, where ε is a predefined thermal equilibrium factor. If the above inequality holds, go to Step 9. Otherwise, update the total route length of the last generation by replacing $Total_{last}$ with $Total_{current}$ if $Total_{current} < Total_{last}$. Increase the *generation_count* by 1. If *generation_count* > *no_gen* go to Step 9 else go to Step 4.

Step 9: If the expected cost of the current solution does not change over the last 20 consecutive temperatures t_i, or a predefined maximum number of temperatures *no_t* have been performed, stop the algorithm and return the current route set and its expected cost as the final solution.

Step 10: Lower the temperature by setting $t_{i+1} = t_i * \alpha$. Set the value of *threshold _infeasible* to be *total_exceed,* the total amount exceeded for the whole route set of the current solution. Set *i* = *i*+1 and go to Step 3.

The steps involved in the TA algorithm only differ from those in SA in the following ways:

1) In Step 2, instead of setting a temperature, we set a threshold value *threshold*.
2) In step 5, we just compare Δ with the threshold value, if Δ < *threshold*, go to Step 6; otherwise, keep the old route set and its expected route length and then go to step 7.
3) The threshold value decreases at the same reduction rate α, e.g., *threshold* = *threshold**α.

4.2.5 Parameter Setting in the SA and TA Algorithms

For the SA algorithm, the parameters are set as follows.

Initial temperature t_0 and temperature reduction parameter α

In this study, the initial temperature t_0 is determined by allowing a certain percentage of deteriorations to be accepted at the beginning of the SA procedure. Suppose the expected cost of the initial solution is $L_{initial}$, and the maximum allowable percentage in cost increase is 10 %, then the initial t_0 can be calculated due to the fact that cost increase as large as $0.1*L_{initial}$ should be accepted with very low probability (such as $prob = 0.001$). Based on the equation $prob = exp(- \Delta /t_0)$, where $\Delta = 0.1*L_{initial}$, we can calculate the t_0. Our computational experience shows that, the maximum allowable cost increase can be set at about 10 % of the expected cost of the initial solution. The temperature reduction parameter α, is set at 0.9.

Final temperature t_f and number of temperatures *no_t*

The final temperature t_f and the total number of temperatures *no_t* are determined in a similar way as described in Section 4.1.2.3.

Thermal equilibrium factor ε and number of generations at each temperature *no_gen*

In this study, we set *no_gen = 20,* and we perform *5*n* transitions within each generation. At different stages of the cooling process, the thermal equilibrium factor ε is set as follows: $\varepsilon = 0.1$ and $\varepsilon = 0.03$ for the first and last 10 temperatures, and $\varepsilon = 0.002$ for the temperatures in between.

Parameters in the TA-infeasibility procedure controlling the infeasibility of the solutions

The initial value of *threshold_infeasible*, *threshold_infeasible0*, is set equal to the total amount exceeded in the initial solution. In later iterations, *threshold_infeasible* only changes when the temperature in SA (or threshold in TA) is lowered, it is set equal to the total amount exceeded in the current solution. Here, we do not decrease the value *threshold_infeasible* steadily as used in other studies, because the function of this TA-infeasibility procedure is to control the infeasibility of the solutions. Moreover, our computational experience shows that, if we decrease *threshold_infeasible* steadily, the search process may have difficulty in finding an acceptable solution in terms of feasibility during the search process, therefore it is trapped in the current solution.

In TA, the parameters are set similar to those in SA, except that, the initial threshold value is set equal to 1 % of the initial route length $L_{initial}$, which means that the initial percentage of the allowable deteriorations is 1 %. We also set the threshold reduction rate $\alpha = 0.9$.

4.3 Computational Results and Analysis

To get the computational results, the algorithms are coded in VISUAL C++, and run on a 500 MHZ Pentium II Processor with 128 MB RAM under Microsoft Windows 98. The problem instances are generated similarly to those used in the comparative study of Chapter 3. All demands D_i are assumed to be identically and independently distributed random variables following discrete uniform distribution.

To get a full picture of how these algorithms perform in different scenarios, the performance of the algorithms is studied when demands follow uniform distribution in the following two cases: with the increase of problem size and with the increase of demand mean and variance.

4.3.1 Single Vehicle Routing Algorithms

Effect of the initial solution on the metaheuristics

For each of the metaheuristics, the search procedure starts from an initial tour. To study the influence of the initial tour, three initial tours: randomly generated tour (RG), tour generated from the nearest neighbor heuristic (NN), and tour generated by the space-filling curve heuristic (SFC), are examined in this study.

Table 4.2 Effect of the initial solution on TS with the increase of problem size

Problem Size (n)	Expected Cost		Computational Time (s)	
	TS-NN	TS-SFC	TS-NN	TS-SFC
20	558.51	570.23	8.4	8.5
40	956.54	946.03	23.6	23.5
60	1301.92	1296.61	56.4	56.5
80	1678.07	1653.43	103.7	105.0
100	2016.31	1995.14	159.1	160.0
120	2347.94	2317.77	235.1	240.4

For TS, we use TS-RG, TS-NN and TS-SFC to denote the TS algorithms starting from a random generated tour, a NN tour and a SFC tour respectively. TS-RG performs very badly. Therefore, we only present the computational results with the increase of problem size and the increase of demand mean and variance (here we use the ratio between the demand mean and the vehicle capacity to represent the increase of demand mean and variance) for TS-NN and TS-SFC as illustrated in Table 4.2 and Table 4.3 respectively. Table 4.2 and Table 4.3 show that, based on the same parameters of TS, except for the case with the smallest problem size ($n = 20$) and the largest demand mean and variance (Mean/Q = 0.4167), solution quality of TS-SFC is better than that of TS-NN. However, their computational time is very close to each other. Therefore, it seems that, TS-SFC behaves better than TS-NN. We select SFC as the initial tour to the TS algorithm in the following computational study.

Table 4.3 Effect of the initial solution on TS with the increase of demand mean and variance

Mean/Q	Expected Cost		Computational Time (s)	
	TS-NN	TS-SFC	TS-NN	TS-SFC
0.0833	929.38	898.26	30.9	31.2
0.1667	1301.92	1296.61	56.5	56.4
0.2500	1713.41	1691.10	82.9	84.1
0.3333	2077.47	2067.81	110.1	111.6
0.4167	2434.70	2446.74	137.7	140.3

However, for the SA and TA algorithms, better initial solution does not necessarily guarantee better final solution. Actually, the random initial tour performs the best. Therefore, the random tour is adopted as the initial solution in the SA and TA algorithms.

Comparison of performances of the metaheuristics

In the following parts, the performance of the proposed metaheuristics is compared against Yang *et al.* (2000)'s single vehicle routing algorithm. Yang *et al.* (2000)'s single vehicle routing algorithm is denoted as Y, and the proposed metaheursitcs are denoted as SA, TA and TS respectively.

Performance with the increase of problem size

We investigate the performance of the algorithms with the increase of problem size when demands follow uniform distribution U [0, 20]. Results are summarized in Table 4.4.

Table 4.4 Comparison of algorithms with the increase of problem size

Problem	Expected Cost				Computational Time (s)			
Size(n)	Y[1]	TA[2]	SA[3]	TS[4]	Y	TA	SA	TS
20	593.85	571.14	577.60	570.23	0.9	3.4	7.4	8.5
40	963.76	982.09	963.46	946.03	6.6	14.7	29.2	23.5
60	1350.36	1378.76	1348.83	1296.61	19.3	42.9	66.9	56.5
80	1700.06	1712.36	1716.64	1653.43	48.6	92.5	116.8	105.0
100	2095.65	2089.47	2061.05	1995.14	90.4	107.7	184.5	160.0
120	2462.10	2439.65	2431.40	2317.77	165.7	155.3	260.5	240.4

[1] Yang *et al.* (2000)'s single vehicle routing algorithm.

[2] Threshold accepting algorithm proposed in this study.

[3] Simulated annealing algorithm proposed in this study.

[4] Tabu search algorithm proposed in this study.

From Table 4.4, we observe that the solution quality of TS performs better than that of the other heuristics in all problem instances studied. As problem size increases, the superiority of TS over the other algorithms also increases, the largest improvement occurs at problem size 120, where the improvement over the SA algorithm (best among the other algorithms) is about 5%. Except for problem size 80, SA performs better than Y; but for TA, it performs better than Y only when problem size is 20 or greater than 80. With respect to the computational time, SA is the most time consuming; though TS takes more computational time than Y, the difference between the times taken by the two algorithms decreases as problem size increases; and TS consumes less time than SA. For TA, it is the least time consuming among the three metaheuristics. It even takes less time than Y at the largest problem size.

Performance with the increase of demand mean and variance

The results for the performance of the algorithms as demand mean and variance increase (with *n* fixed at 60) is summarized in Table 4.5. In terms of the solution quality, over the whole demand range, TS outperforms the other algorithms. Compared with SA (which is much better than both Y and TA), the improvement of TS over SA decreases as demand mean and variance increase, with the largest improvement being 5%. With respect to computational time, the computational times taken by SA, TA and TS are several times longer than that of Y. Nevertheless, TS is less time consuming than SA, and TA is the least time consuming one among SA, TA and TS.

Table 4.5 Comparison of algorithms with the increase of demand mean and variance

Mean/	Expected Cost				Computational Time (s)			
Q*	Y[1]	TA[2]	SA[3]	TS[4]	Y	TA	SA	TS
0.0833	956.09	950.70	944.34	898.26	12.5	45.3	40.9	31.2
0.1667	1350.36	1378.76	1347.07	1296.61	19.3	42.9	65.2	56.4
0.2500	1743.54	1753.67	1756.67	1691.10	27.0	56.0	90.7	84.1
0.3333	2140.18	2133.33	2104.54	2067.81	33.5	70.8	120.1	111.6
0.4167	2532.53	2480.88	2472.30	2446.74	37.74	69.9	149.1	140.3

* The ratio between the demand mean and the vehicle capacity, which represents the increase of demand mean and variance.

[1] Yang *et al.* (2000)'s single vehicle routing algorithm.

[2] Threshold accepting algorithm proposed in this study.

[3] Simulated annealing algorithm proposed in this study.

[4] Tabu search algorithm proposed in this study.

4.3.2 Multiple Vehicle Routing Algorithms

In Yang *et al.* (2000), two algorithms were presented: one is route-first-cluster-second (denoted as R1-C2) and the other is cluster-first-route-second (denoted as C1-R2). In this section, the performance of the proposed SA algorithm (denoted as SA) and TA algorithm (denoted as TA) is compared with Yang *et al.* (2000)'s algorithms based on the solution quality, computational time, and the number of vehicles required.

Performance with the increase of problem size

Here, we assume that all demands are generated from the discrete uniform distribution U [0, 30]. The solution quality, computational time and the number of vehicles required of the algorithms with the increase of problem size are illustrated in Table 4.6.

In terms of solution quality, it is clear that performances of SA and TA are very close to each other, and they are both superior to R1-C2 and C1-R2, except when the problem size is 40, TA is slightly worse than R1-C2. As problem size increases, the superiority of TA and SA over R1-C2 and C1-R2 becomes more obvious, which indicates that both TA and SA are capable of finding solutions with better quality for large scale problems.

With respect to computational time, C1-R2 is the least time consuming algorithm. For the other three algorithms, TA takes less time than SA. Computational time of R1-C2 increases much faster than the other algorithms. Therefore, when problem size is less than 100, R1-C2 consumes the least time among the three algorithms. However, when the problem size ranges from 100 to 120, R1-C2 takes more time than TA but less time than SA. When the problem size becomes even larger, computational time of R1-C2 exceeds both that of TA and SA. In terms of

Table 4.6 Average performance with the increase of problem size

Problem	Expected Cost				Computational Time (s)				Number of Vehicles			
Size (n)	SA[1]	TA[2]	R1-C2[3]	C1-R2[4]	SA	TA	R1-C2	C1-R2	SA	TA	R1-C2	C1-R2
40	1260.85	1272.65	1263.57	1318.88	69	31	11	7	5.5	6.2	5.7	5.5
60	1775.74	1795.12	1802.45	1867.14	134	67	39	15	7.4	8.1	8.0	7.6
80	2291.34	2299.28	2331.79	2428.95	214	114	87	30	9.4	9.9	10.2	9.5
100	2777.29	2783.54	2835.98	2942.58	258	143	184	50	11.5	11.8	12.6	11.4
120	3280.64	3283.95	3344.51	3445.79	343	228	346	74	13.7	14.0	14.7	13.3
140	3784.77	3787.28	3858.74	3963.67	347	244	554	126	16.1	16.0	16.8	15.1

[1] The simulated annealing algorithm proposed by this study for the VRPSD

[2] The Tabu search algorithm proposed by this study for the VRPSD

[3] Yang *et al.* (2000)'s route-first-cluster-second algorithm.

[4] Yang *et al.* (2000)'s cluster-first-route-second algorithm.

computational time, TA and SA also have potential for large scale problems. For the number of vehicles required, both R1-C2 and TA require more vehicles than C1-R2 and SA. Comparing C1-R2 and SA, results show that, when the problem size is less than 100, SA requires less number of vehicles; however, when problem size becomes larger, C1-R2 uses less number of vehicles.

Performance with the increase of demand mean and variance

Table 4.7 shows how the algorithms perform with the increase of demand mean and variance in terms of solution quality, computational time and the number of vehicles required. Here, the problem size is fixed at 60 nodes. Here we use the ratio between the demand mean and the vehicle capacity (MEAN/Q) to represent the increase of demand mean and variance.

Comparing the solution quality of the 4 algorithms, C1-R2 is obviously the worst. For the other three algorithms, when demand mean is relatively low (less than ¼ of the capacity *Q*), their performances are quite similar. However, as demand mean further increases, both SA and TA start to outperform R1-C2; meanwhile, though solution quality of TA and SA is very close to each other, SA behaves a little better than TA. The computational time in Table 4.7 indicates that, as demand mean increases, the computational time does not increase significantly for all of the algorithms except for R1-C2. Among the four algorithms examined, C1-R2 takes the least time; while SA takes the most time, followed by TA and R1-C2. For the number of vehicles, Table 4.7 illustrates that, when demand mean is relatively low (less than ¼ of the capacity *Q*), TA requires the most number of vehicles, while C1-R2 requires the least; however, as demand mean increases, SA requires the least number of vehicles, and R1-C2 gradually becomes the one requires the most number of vehicles.

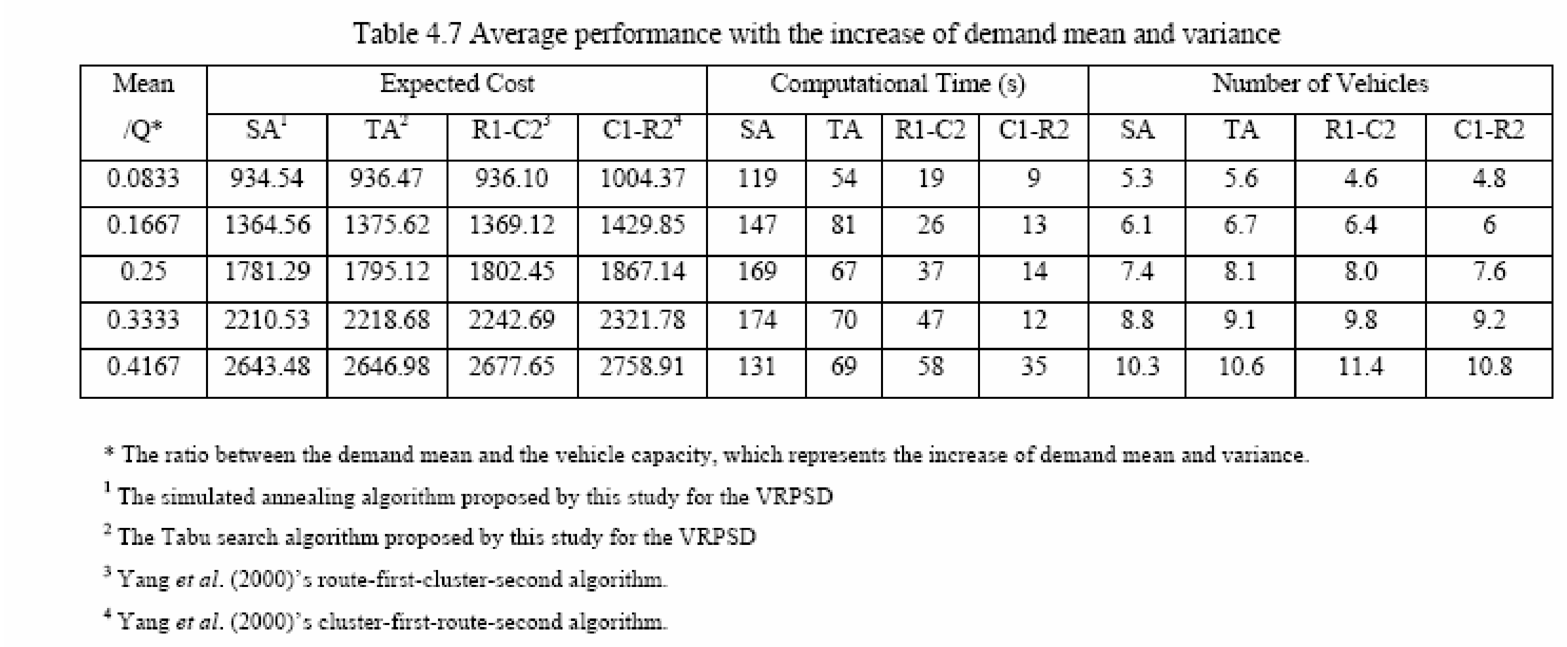


Table 4.7 Average performance with the increase of demand mean and variance

Mean	Expected Cost				Computational Time (s)				Number of Vehicles			
/Q*	SA[1]	TA[2]	R1-C2[3]	C1-R2[4]	SA	TA	R1-C2	C1-R2	SA	TA	R1-C2	C1-R2
0.0833	934.54	936.47	936.10	1004.37	119	54	19	9	5.3	5.6	4.6	4.8
0.1667	1364.56	1375.62	1369.12	1429.85	147	81	26	13	6.1	6.7	6.4	6
0.25	1781.29	1795.12	1802.45	1867.14	169	67	37	14	7.4	8.1	8.0	7.6
0.3333	2210.53	2218.68	2242.69	2321.78	174	70	47	12	8.8	9.1	9.8	9.2
0.4167	2643.48	2646.98	2677.65	2758.91	131	69	58	35	10.3	10.6	11.4	10.8

* The ratio between the demand mean and the vehicle capacity, which represents the increase of demand mean and variance.

[1] The simulated annealing algorithm proposed by this study for the VRPSD

[2] The Tabu search algorithm proposed by this study for the VRPSD

[3] Yang *et al.* (2000)'s route-first-cluster-second algorithm.

[4] Yang *et al.* (2000)'s cluster-first-route-second algorithm.

Comments on the computational results

Computational results discussed above demonstrate that, both TA and SA are capable of finding solutions with better quality than both R1-C2 and C1-R2. This is quite reasonable, because the heuristic procedures employed in R1-C2 and C1-R2 are classical local search algorithms, which have the tendency to be trapped in the local optimum. While both SA and TA are metaheuristics in the sense that they are capable of escaping from local optimum and finally obtain a near global optimum solution.

Computational results also show that, TA is a less time consuming algorithm than SA. This may be due to the following reasons:

The first reason is related to the feature of TA and SA. In SA, to escape from the local optimum, it accepts deteriorations in objective function value with a certain probability, which is calculated by using exponential expressions; while in TA, whether a solution with deteriorations in objective function value is accepted is determined by comparing this deterioration with a threshold value (a constant in a certain context). Obviously, the later one is more simple and with less computational complexity.

The second reason is related to the parameter setting in the TA and SA algorithms. In TA, the initial threshold value is set to 1 % of the total cost of the initial solution, which means that the algorithm can at most accept a solution with deteriorations equal to this threshold value. In SA, to get similar solution quality, the initial temperature must be high enough to at most accept a solution with deteriorations equal to about 10 % of the total cost of the initial solution. Obviously, the later one is more time consuming.

4.4 Conclusions

In this chapter, we studied how metaheuristics perform for the VRPSD. We present three heuristics: SA, TA and TS for the single VRPSD; and two heuristics SA and TA for the multiple VRPSD.

For the single vehicle case, we examine the effect of different initial solutions on the performance of the algorithms, and carry out a comparative study to see how metaheuristics perform in comparison with other heuristics presented in the literature. Results show that, for the TS algorithm, the better the initial solution, the better the final solution; however, for the SA and TA, better initial solution does not necessarily guarantee better final solution. The comparative study shows that, the solution quality of the TS outperforms the other three heuristics with the increase of both problem size and the increase of demand mean and variance. With respect to computational time, metaheuristics are much more time consuming. However, when comparing TS with SA, it takes less computational time. Though TA is the least time consuming one among the three metaheuristics, its solution quality is not quite good, it even performs worse than Yang *et al.* (2000)'s algorithm in some cases. Since all three metaheuristics employ the same neighborhood structure, we can conclude that, in terms of both solution quality and computational time, the TS algorithm performs better than the SA and TA algorithms for the single VRPSD.

For the multiple vehicle case, our study reveals that, solution quality of TA and SA is very close to each other, and both SA and TA can provide solutions with less expected cost than the two algorithms presented in Yang *et al.* (2000) for almost all of the problems tested. With respect to computational time, both SA and TA are more time consuming for the small and medium sized problems. However, they take less time than R1-C2 for problems with 120 or more nodes. These results lead to the conclusion that, in terms of both solution quality and computational time, SA and TA are more suitable for large size problems. One more finding is that, computational time of TA is less than that of SA. Since their solution quality is similar, TA is more preferable to be applied in practical applications.

Chapter 5

Algorithms for the Multi-period TCTSP in a Rolling Schedule Environment

In this chapter, we consider the multi-period TCTSP in a rolling schedule environment. The problems considered and the frameworks of this study are described in Section 5.1. Section 5.2 presents a set-covering type formulation for the problem within one rolling horizon. Sections 5.3 and 5.4 describe several heuristic algorithms based on iterative customer assignment (ICA) and the iterative center-of-gravity (ICG) scheme. In Section 5.5, the set-covering formulation presented in Section 5.2 is considered within one elongated rolling horizon, and solved to optimality as a linear programming (LP) problem by a column generation scheme to get an upper bound. Meanwhile, based on the final basis, a feasible integer solution is obtained to examine the gap between the LP relaxation solution and an integer solution. Section 5.6 discusses the computational results and performances of the heuristics.

5.1 Problem Description and Framework of the Study

The problem considered in our study is a multi-period TCTSP in a rolling schedule environment. We now briefly describe the problem as follows:

In each period, customers may call a company for services. In a call *j*, the customer specifies a date g_j and a time tolerance σ_j, which means that the company should visit the customer within the time window $g_j \pm \sigma_j$. If the service can be made within the time period $g_j \pm \sigma_j$, a profit of R_j realizes; otherwise the profit is zero. Assume the rolling horizon has a length of H periods, that is, for those customers who called before the current period, if their possible visit periods required fall into the H periods, they are eligible to be scheduled in this rolling horizon. The problem is to find a subset of customers as well as the sequence of serving them in each period in a rolling schedule environment, so that the average profit per period in the long run is maximized.

Regarding the problem considered in this study, there is one thing in common with the TSSP+1 (a traveling salesman subset-tour problem with an additional constraint) class of problems discussed in Mittenthal and Noon (1992): not only do we need to figure out the order in which to visit the customers, but we must answer the more fundamental question: which customers do we want to visit? However, there are also some differences: the time window and the dynamically changing customer information. It is these differences that make us consider the problem from a rolling schedule point of view. Firstly, since a customer may be scheduled before or after its desired period of visit provided that the visit is within the time window, if we know the customer demand information for a certain number of periods, we can make schedules for these periods rather than making a schedule only for one period. The rational is as follows: Suppose that for the current period, there are many customers eligible for visiting. Some customers are less profitable and the current period is at the end of their time windows; while some customers are more profitable and the current period is at the beginning of their time windows. If we only consider maximizing profit of the current period, the later customers may be selected and the former ones may be dropped. However, if the number of customers requesting services at the next period happens to be small, obviously we lost the profits that may be realized from the dropped customers. Making schedules for several periods rather than only one period can avoid this kind of myopia and therefore maximize the average profit in the long run. On the other hand, since in each period, new customers may randomly require services, customer information is not static but dynamically changing from period to period. As a result, the customer information is only accurate for the current period; and the information accuracy decreases as the future period goes farther from the current period. Therefore, we can only implement the schedule for the current period, as customer information for the later periods is not accurate. After implementing the schedule for the current period, we update the data according to new information available. We then consider a new rolling horizon by forwarding the time period by one period and find a schedule for the new rolling horizon. This rolling schedule scheme nicely captures the characteristic of the problem considered in this study.

5.2 A Set-covering Type Formulation

In the rolling schedule environment, it is very difficult, if not impossible, to mathematically formulate the problem, due to the dynamically changing customer information from one rolling horizon to the next. Therefore, we consider one rolling horizon with *H* periods, and present a set-covering type formulation within one rolling horizon for the problem.

The set-covering problem has been studied in great detail since the mid 1960s. Among many of its applications, some representative ones described in the literature include: airline fleet scheduling (Levin, 1969), truck routing (Balinski and Quandt, 1964; Clark and Wright, 1964; Dantzig and Ramser, 1959; Garfinkel and Nemhauser, 1969), airline crew scheduling (Arabeyre *et al.*, 1969; Spitzer, 1961), stock cutting

(Pierce, 1970), assembly line balancing (Salveson, 1955), facilities location problems (Revelle *et al.*, 1970), and coloring problems (Busacker and Saaty, 1965).

The following notations are used in the description of the formulation. Associated with each customer *j*, given the date g_j and time tolerance σ_j that the customer requires service, we define $[e_j, l_j]$ as the time window the service should be provided, with $e_j = g_j - \sigma_j$ and $l_j = g_j + \sigma_j$. Let [1, *H*] represent the rolling horizon with *H* periods, and *V* the set of all customers in the rolling horizon [1, *H*], such that $[e_j, l_j] \cap [1, H] \neq \phi$. That is, $V = \{j \mid j$ is a customer and $[e_j, l_j] \cap [1, H] \neq \phi\}$.

Let Ω_i be the set of all possible sub-tours in period *i*.

Define: $x_k^i = \begin{cases} 1 & \text{if } k^{th} \text{ possible sub-tour is chosen in period } i \\ 0 & \text{otherwise} \end{cases}$

Suppose that totally *n* customers are eligible for service in the rolling horizon. Let $\underset{\sim}{a_k^i}$ be a vector of size *n* (composed of zeros and ones), representing *k*th sub-tour in period *i*, and be defined in such a way that a_{jk}^i = 1, if customer *j* is included in *k*th sub-tour in period *i*, 0 otherwise. All possible sub-tours in set Ω_i can be generated by taking into consideration the prevailing conditions and constraints imposed on the assignment of customers to the sub-tours. One important consideration for this problem is that a customer *j* must be assigned to the periods within its time window $[e_j, l_j]$. Another consideration is that, the total travel and service time of each sub-tour $\underset{\sim}{a_k^i}$ must be within the effective working time of a salesman in period *i*.

Let R_k^i represent the sales revenue if customers are served in *k*th sub-tour in period *i*. Then a mathematical programming model of this problem for the finite one rolling horizon case can be formulated as follows:

$$Max\ Z = \sum_{i=1}^{H} \sum_{k \in \Omega_i} R_k^i x_k^i \Big/ H \tag{5.1}$$

subject to

$$\sum_{k \in \Omega_i} x_k^i = 1 \quad i = 1, 2, \ldots, H \tag{5.2}$$

$$\sum_{i=1}^{H} \sum_{k \in \Omega_i} a_{jk}^i x_k^i \leq 1 \quad \text{for all } j \in V \tag{5.3}$$

x_k^i = 0, 1 for all *i* and *k*.

Here, we denote this formulation as ***Problem P***.

Constraints (5.2) indicate that, in any period *i* = 1, 2,…, *H*, among all the possible sub-tours in Ω_i , exactly one sub-tour must be chosen. Constraints (5.3) suggest that for all the customers $j \in V$, within the rolling horizon, they can be visited at most once.

5.3 Solution Method Based on Iterative Customer Assignment (ICA) Scheme

To solve the set-covering type formulation presented in Section 5.2, one possible way is to employ column generation scheme. However, in the rolling schedule environment, the problem has to be solved many times; column generation method may be time consuming. Therefore, we need a more efficient way to solve the problem. We present the following iterative customer assignment (ICA) procedure as follows.

The iterative customer assignment scheme includes two procedures: one assigning procedure and one routing procedure. It is very similar to the cluster-first-route-second method, where the clustering procedure corresponds to the assigning procedure here. The difference is that, once the routing procedure results in possibilities of serving other customers in certain periods, the assigning procedure is called again, and the assigning and routing procedures are iterated until no more customers can be assigned to any of the periods.

To facilitate the description in the later sections, we give the additional denotations needed as follows:

p_{ij} = Profit of customer *j* if it is assigned to period *i*; $j \in V$, $i = 1, 2, ..., H$.

w_{ij} = Weight of customer *j* if it is assigned to period *i*. It is the additional traveling time plus service time needed in period *i*, which is dynamic and dependent on the certain sub-tour that has been constructed before customer *j* is added to period *i*.

T = The effective working time for each period.

T_i = The remaining working time of period *i*.

The assigning procedure is to select *H* subsets of customers so that the total profit of the selected customers is maximized, and the total weight of all customers assigned to each period is no more than *T*. Several heuristic procedures for the assigning procedure are proposed in Section 5.3.2.

The routing procedure mainly tries to decrease the traveling time of the route in each period, since the revenue of visiting a customer is the same in different period within the time window. We first apply 2-opt (Croes, 1958) to improve the performance of the tour in each period. Then, try to further decrease the traveling time of the route by performing swaps between two customers in different period, provided that the swap is feasible with respect to the time window and time limit constraint.

5.3.1 ICA Procedure

The procedure involved in the ICA can be described as follows:

Step 1: Redefine the time window of customer j, $[e_j, l_j]$, with $e_j = \max\{e_j, 1\}$ and $l_j = \min\{H, l_j\}$.

Step 2: Assign the customers in V to the most appropriate periods by some heuristic methods described in Section 5.3.2.

Step 3: Perform 2-opt and swaps of customers to improve the tour of each period.

Step 4: Repeat Steps 2 and 3 until no more customers in set V can be assigned to any of the periods.

Step 5: Implement the routing sequence of customers obtained in Step 4 for the first period. Update the data according to new information available. Now consider a new rolling horizon by forwarding the time period by one period. Repeat the whole procedure again to find a schedule for the new rolling horizon.

5.3.2 Heuristics for the Assigning Procedure

To assign the customers to the most appropriate periods, one prerequisite is the construction of weight matrix (w_{ij}). It is constructed as follows: Given a set of sub-tours constructed for periods $i = 1, 2, \ldots, H$, we try to insert a customer $j \in V$ in each of the sub-tours within the time window $[e_j, l_j]$. The weight w_{ij} is defined as the additional traveling time plus the service time, if customer j is to be added to the sub-tour of period i. If a customer j cannot be feasibly added to the sub-tour in period i due to the time window constraints, w_{ij} is set to $+\infty$. The value of the weight w_{ij} calculated in this way is not exactly the additional time spent when a customer is visited in a certain period, because the route in each period keeps on changing during the assigning procedure.

Several heuristics for the assigning procedure are proposed in this study. They are described as follows.

Heuristic algorithm 1 (HA1) - Assign customers based on its weight and the remaining working time

HA1 procedure

Step 1: Rank each customer $j \in V$ in descending order of its revenue R_j.

Step 2: Start from the top of the list, select a customer j and calculate w_{ij}. Try to assign it to the period i^* with the largest ratio of T_i/w_{ij} (i = 1, 2,…, H). If infeasible, go to Step 4.

Step 3: Insert the selected customer j into the tour of period i^* in the cheapest way, update T_i, the remaining working time of period i^*.

Step 4: Remove customer j from the set V. Go to Step 2 and continue the assigning procedure until no more customers can be assigned to any period.

Heuristic algorithm 2 (HA2) – Assign customers by choosing the most profitable period for each customer

Martello and Toth (1981) presented a polynomial-time heuristic algorithm for the generalized assignment problem. Let f_{ij} be a measure of the desirability of assigning item j into knapsack i ($i = 1, 2, ...K$). Iteratively consider all the unassigned items, and determine the item j* having the maximum difference between the largest and the second largest f_{ij}; j^* is then assigned to the knapsack for which f_{ij^*} is a maximum. The above algorithm assigns items to the knapsacks based on opportunity cost. In our problem, similarly, each customer can be assigned to one of several periods, we need to find the most appropriate period for this customer; however, unlike the above case, some customers can be left over to be considered in the next rolling horizon.

To apply the Martello and Toth (1981) algorithm to our problem, we need to discuss the construction of the profit matrix p_{ij} first. In the rolling schedule environment, we are more concerned with the profit generated from the first period, because firstly, we only implement the decisions of the first period; secondly, customer information for the later periods may be subject to some changes. Therefore, when solving this problem, we tried three methods to construct the profit matrix to see which one is more suitable in a certain situation.

R1: p_{ij} based on actual revenue R_j of visiting a customer j. The p_{ij} of the profit matrix with H rows and n columns is defined as follows:

$$p_{ij} = \begin{cases} R_j, & i \in [e_j, l_j] \\ 0, & \text{otherwise} \end{cases}$$

R2: p_{ij} based on artificial revenue R_j of visiting customer j by putting more weight on assigning customer j to the early period of its time window. As a result, customers tend to be visited during the early periods of the time window.

$$p_{ij} = \begin{cases} R_j - R_j\left(i - \frac{e_j + l_j}{2}\right)/(l_j - e_j), & i \in [e_j, l_j] \\ 0, & \text{otherwise} \end{cases}$$

R3: p_{ij} based on artificial revenue R_j of visiting customer j within the desired period of service. In this case, for a call j with specified date g_j and time tolerance σ_j ($e_j = g_j - \sigma_j$ and $l_j = g_j + \sigma_j$), the service would most likely be provided at period g_j.

$$p_{ij} = \begin{cases} R_j - R_j(|i - \frac{e_j + l_j}{2}|)/(l_j - e_j), & i \in [e_j, l_j] \\ 0, & \text{otherwise} \end{cases}$$

In this study, for the measure of the desirability f_{ij}, we examine three cases:

M1: $f_{ij} = p_{ij}$ (5.4)

M2: $f_{ij} = p_{ij} / w_{ij}$ (5.5)

M3: $f_{ij} = T_i \, p_{ij} / w_{ij}$ (5.6)

The heuristic applying these measures is given below.

HA2 procedure

Step 1: Compute the weight matrix (w_{ij}) and value f_{ij} for each customer $j \in V$.

Step 2: For each customer $j \in V$, rank the periods in descending order in terms of f_{ij} ($i = 1, 2, ..., H$). Go to Step 3a if we assign based on opportunity desirability; go to Step 3b if we assign based on actual desirability.

Step 3a: For each customer $j \in V$, find the first two periods on the top of the list; calculate the difference between the desirability f_{ij} of these two periods, called opportunity desirability of customer j. Among all customers in V, find the customer j^* and the period i^* with the largest opportunity desirability. Go to Step 4.

Step 3b: Among all customers in V, find the customer j^* and the period i^* with the largest desirability f_{ij}.

Step 4: Test whether it is feasible to insert customer j^* into the tour of period i^* in a cheapest way. If the insertion is infeasible with respect to the time limit constraint, go to Step 1. Otherwise, insert customer j^* into the tour of period i^* in a cheapest way, update the remaining capacity T_i of period i^*, remove customer j^* from V and go to Step 1.

Step 5: Repeat Step 1 to Step 4 until no more customers can be assigned to any of the periods.

Heuristic algorithm 3 (HA3) – Assign customers based on profitability

In HA2, we choose the most appropriate period for each customer. In contrast, the rational of heuristic HA3 is as follows: for each period, there are a number of eligible customers, choose the most profitable customers for each period. We also apply f_{ij} to measure how desirable a customer j is assigned to period i and employ the three ways (**M1**, **M2** and **M3**) to calculate f_{ij}. However, here we only need to calculate p_{ij} based on the method **R1** as described above.

The procedure for heuristic HA3 is the same as that of HA2, except that Steps 2, 3a and 3b are replaced by the following three steps:

Step 2: For each period i, find all possible customers $j \in V$ that can be assigned to this period. Rank the customers in period i in descending order in terms of the desirability f_{ij}. Go to Step 3a if we assign based on opportunity desirability; go to Step 3b if we assign based on actual desirability.

Step 3a: For each period i, find the first two customers on the top of the list, calculate the difference between the desirability f_{ij} of these two customers, call it opportunity desirability of the first customer. Among all periods, find the customer j^* and the period i^* with the largest opportunity desirability. Go to Step 4.

Step 3b: Consider the customer on the top of the list for each period, find the customer j^* and the period i^* with the largest desirability f_{ij} among all periods. Go to Step 4.

5.4 Solution Method Based on Iterative Center-of-Gravity (ICG) Scheme

The core concept of the iterative center-of-gravity (ICG) scheme is the center-of-gravity. Suppose that node j has coordinates (x_j, y_j) and revenue R_j, and the tour of period i is denoted as π_i, the center-of-gravity of π_i, $G_i(x, y)$, can be calculated as follows:

$$x = \sum_{j\in\pi_i} x_j R_j \Big/ \sum_{j\in\pi_i} R_j \tag{5.7}$$

$$y = \sum_{j\in\pi_i} y_j R_j \Big/ \sum_{j\in\pi_i} R_j \tag{5.8}$$

In addition to the denotations introduced in Section 5.3, we further suppose that,

d_{ij} = distance between customer j and the center-of-gravity of π_i, $G_i(x, y)$.

Heuristic algorithm 4 (HA4) – Assignment based on ICG

The procedure involved in the ICG scheme can be described as follows:

HA4 procedure

Step 1: Redefine the time window of customer j, [e_j, l_j], with e_j = max{e_j, 1} and l_j = min{H, l_j}. Form the initial tour π_i of each period i by using those customers who request services at period i with σ_i equals to zero. If all σ_i are greater than zero, tour π_i is initialized to contain the depot only.

Step 2: Calculate the center-of-gravity $G_i(x, y)$ for the tour π_i of each period i.

Step 3: For each customer j eligible to be visited in period i with respect to its time window, calculate d_{ij}, the distance between customer j and center-of-gravity of the tour of period i.

Step 4: Assign customer j to period i according to one of the following 3 criteria:

C1: Assign customer j to the earliest period of its time window.

C2: Assign customer j to period i with the largest value of p_{ij}/d_{ij}

C3: Assign customer j to period i randomly satisfying the condition that customer j has more chances to be visited in a period with larger value of p_{ij}/d_{ij}.

Step 5: Rank the customers assigned to each period i in descending order of the value p_{ij}/d_{ij}.

Step 6: For each period, starting from the top of the list, insert customers based on cheapest insertion heuristic, until no more customers can be inserted without exceeding the time limit.

Step 7: Perform 2-opt within each route and swap customers between routes (similar to the one described for ICA procedure in Section 5.3.1) to further decrease the traveling time of the routes. For the remaining (not yet assigned) customers eligible to be visited in period i, insert them to the tour by using the cheapest insertion heuristic until no more customers can be inserted into the tour without violating the time limit constraint.

Step 8: Repeat Step 2 to Step 7 until the profit realized from visiting the customers does not change much or a predefined number of iterations have been executed.

Step 9: Implement the routing sequence of customers obtained in Step 8 for the first period. Remove the customers scheduled in the first planning period in the set *V*. Update the data according to new information available. Forward the time period by one period and repeat the whole procedure again to find a schedule for the new rolling horizon.

To examine the effect of the artificial profit p_{ij}, we also try three ways (**R1**, **R2** and **R3**) to construct the matrix (p_{ij}) as described in Section 5.3.2.

5.5 An Upper Bound Generated Based on the Set-covering Type Formulation and Column Generation Solution Method

In Section 5.2, we present a set-covering type formulation for the problem within one rolling horizon. In this section, we generate an upper bound for the problem based on this formulation by making the following assumptions.

- The rolling horizon is elongated in such a way that it covers all periods considered in each of the heuristics.
- Customer demand information is known for all the periods considered when we calculate the upper bound.

For solving the set-covering problem, there are different procedures presented in the literature. These include: implicit enumeration (Balas, 1965; Pierce, 1968; Garfinkel and Nemhauser, 1969), heuristics (Christofides, 1974; Baker *et al.*, 1979), simplex based cutting plane methods (Gomory, 1963), hybrid primal cutting plane/implicit enumeration method (Balas, 1975), set partitioning via node covering (Balas and Samuelson, 1974), network flow model (Moreland, 1966), and column generation algorithm (Balas and Padberg, 1975).

Column generation is an effective and commonly used method for designing algorithms for the problems formulated as set-covering type model. Therefore, in this study, we consider the column generation method for the set-covering model of the multi-period TCTSP. From the LP relaxation of the set-covering model, it is clear that, if all possible sub-sets can be enumerated, one can obtain the optimal solution for the LP. However, the set of possible sub-sets can be very large for large size problems. Therefore enumeration is impractical. By applying column generation scheme, it is not necessary to explicitly enumerate all feasible sub-sets; one can find the non-basic variable with the largest non-negative reduced cost by solving an optimization problem, called the pricing problem. Successful applications of column generation in this kind of problems can be found in the literature: Staff scheduling (Sarin and Aggarwal, 2001), optimal shift scheduling (Mehrotra *et al.*, 2000), the general pickup and delivery problem (Savelsbergh and Sol, 1998), etc.

5.5.1 **Column Generation Scheme**

Consider the ***Problem P*** formulated in Section 5.2. Let ***P’*** denote the LP relaxation of ***Problem P***, it is clear that, if we can enumerate all possible sub-tours in set Ω_i of period *i*, we can obtain the optimal solution of the problem ***P’*** by solving a LP problem. However, the set of possible sub-tours Ω_i ($i=1, 2,..., H$) can be very large for large size problems. Therefore a column generation scheme is adopted to find the non-basic variable with the largest non-negative reduced cost by solving an optimization problem, called the pricing problem.

Since problem ***P'*** is an LP relaxation of ***Problem P***, the optimal solution to problem ***P'*** serves as an upper bound for ***Problem P***. This upper bound will be employed as a performance measurer in the computational experiment of Section 5.6.2.

The master problem

Let ***P'*** be the LP relaxation of the ***Problem P***. Suppose that for each period $i = 1$, 2, …, H, a set $\Omega_i' \subseteq \Omega_i$ of feasible sub-tours is explicitly known, we can define the restricted master problem of ***P'*** as follows:

$$Max\ Z = \sum_{i=1}^{H} \sum_{k \in \Omega_i'} R_k^i x_k^i \Big/ H \tag{5.9}$$

subject to

$$\sum_{k \in \Omega_i'} x_k^i = 1 \quad i = 1, 2, \dots, H \tag{5.10}$$

$$\sum_{i=1}^{H} \sum_{k \in \Omega_i'} a_{jk}^i x_k^i \le 1 \quad \text{for all } j \in V \tag{5.11}$$

$$x_k^i \ge 0 \tag{5.12}$$

Suppose that the restricted master problem of ***P'*** has a feasible solution x, and let (o, u) be the associated dual solution, i.e., dual variables o_i and u_j are associated with the constraints (5.10) and (5.11) respectively. From linear programming duality we know that x is optimal with respect to ***P'*** if and only if for each $i = 1, 2, \dots, H$, and for each $k \in \Omega_i$, the reduced cost d_k^i is non-positive. That is,

$$d_k^i = R_k^i - \sum_{j \in V} a_{jk}^i u_j - o_i \le 0 \text{ for all } i = 1, 2, \dots, H,\ k \in \Omega_i \tag{5.13}$$

Therefore, to test whether *x* is optimal with respect to ***P'***, we can solve the following pricing problem:

$$Max \left\{ R_k^i - \sum_{j \in V} a_{jk}^i u_j - o_i \mid i = 1, 2, \dots H,\ k \in \Omega_i \right\} \tag{5.14}$$

Suppose we use ***z*** $\boldsymbol{z_{price}}$ to denote the value of the solution to the above pricing problem, and let I_z and K_z represent the corresponding period and route. If $z_{price} \le 0$, then *x* is optimal with respect to ***P'***; otherwise, K_z define a column that can enter the basis and has to be added to Ω_{I_z}'. So the column generation procedure can be illustrated as follows:

Step 1: Find the initial sets $\Omega_i' \subseteq \Omega_i$ containing a feasible solution *x*.

Step 2: Solve the restricted master problem of ***P'***.

Step 3: Solve the pricing problem. If $z_{price} \le 0$, stop; otherwise, set $\Omega_{I_z}' = \Omega_{I_z}' \cup \{K_z\}$ and go to Step 2.

In our problem, the initial sets $\Omega_i' \subseteq \Omega_i$ containing a feasible solution to ***P'*** can be found by using any of the heuristics developed in Sections 5.3 and 5.4.

The pricing problem

The pricing problem above is to find the most profitable sub-tour among all sub-tours for all periods considered. The problem can be decomposed into several independent problems, one for each period, since

$$Z_{price} = Max \left\{ R_k^i - \sum_{j \in V} a_{jk}^i u_j - o_i \mid i = 1, 2, \dots, H, k \in \Omega_i \right\} \tag{5.15}$$

is equivalent to

$$Z_{price} = \underset{i=1,2,...H}{Max} \ \text{Max}\left\{ R_k^i - \sum_{j \in V} a^i_{jk} u_j - o_i \mid k \in \Omega_i \right\} \tag{5.16}$$

Thus, the pricing problem for period *i* can be expressed as

$$Z^i_{price} = \text{Max}\left\{ R_k^i - \sum_{j \in V} a^i_{jk} u_j - o_i \mid k \in \Omega_i \right\} \tag{5.17}$$

which tries to find the most profitable sub-tour for period *i* that serves a subset of the customers.

The column selection

From the literature, there are several ways to select the columns to add to the restricted master problem. One is to select the column with the maximum reduced cost; the other is to select some columns with positive reduced cost. In the former case, the LP problem will not grow very rapidly, but it has to be solved for each column added. While in the later case, the number of LP problems that have to be solved will be reduced, but the LP solved each time will become very large. In our problem, we select and add *H* columns one for each period (pricing problem) to the master problem before we solve it.

Obtaining an integer solution

The solution to the ***P'*** problem serves as an upper bound of the ***Problem P***. To examine whether this upper bound is tight, we can examine how far away this bound is from a good feasible integer solution. This feasible integer solution can be obtained by applying CPLEX mix-integer optimizer based on the final basis of problem ***P'***.

5.5.2 Solving the Pricing Problem

From the above, we know that our pricing problem is

$$Z_{price} = \underset{i=1,2,\ldots H}{Max}\ \text{Max}\left\{R_k^i - \sum_{j\in V} a_{jk}^i u_j - o_i \mid k \in \Omega_i\right\} \tag{5.18}$$

By making some manipulations, the pricing problem can become a knapsack problem for each period *i* as shown below.

Given that R_j is the profit of serving customer *j*, the total profit of visiting all customers in route *k* of period *i*, R_k^i, can be written as:

$$R_k^i = \sum_{j\in V} a_{jk}^i R_j \tag{5.19}$$

The pricing problem becomes,

$$Z_{price} = \underset{i=1,2,\ldots H}{Max}\ \text{Max}\left\{\sum_{j\in V} a_{jk}^i (R_j - u_j) - o_i \mid k \in \Omega_i\right\} \tag{5.20}$$

subject to

$$\sum_{j\in V} a_{jk}^i w_{jk}^i \le T \tag{5.21}$$

where w_{jk}^i is the additional traveling time and service time of visiting customer *j* in route *k* of period *i*.

One more thing to consider in the pricing problem is the time window constraint for each customer *j*. We do not embody this constraint in the master problem. We can deal with it here by defining $w_{jk}^i = \infty$, when we try to serve a customer in the period outside its time window. Then, the problem becomes: for each period, first we decide which customers can be visited in this period in terms of time window constraint; then select a subset of the customers that maximize the net profit while satisfying the time limit constraint. Since weight w_{jk}^i in the pricing problem is not a constant as in real knapsack problems, the pricing problem is actually a TCTSP or an orienteering problem with starting and ending point at the depot.

In the column generation scheme, if we want to solve the master problem optimally, we have to solve the pricing problem to optimality. However, to save computational time, we first solve the orienteering problem with a heuristic based on center-of-gravity method (Golden *et al.*, 1987). We add one column for each period to the restricted master problem until the heuristic can not discover any more columns with positive reduced cost. Then the orienteering problem is solved to optimality to further search the columns with positive reduced cost.

To solve the pricing problem to optimality, we need to present a mathematical formulation for the problem. Based on the following notations, an integer programming model for the pricing problem in one period *i* is presented below.

V_i = A set of customers that can be visited in period *i*. That is, $V_i = \{j \mid j$ is a customer and $e_j \le i \le l_j\}$.

$V_i^{'}$ = $V_i \cup \{0\}$.

A = $\{(d, j) \mid d, j \in V_i^{'}$ and $d < j\}$ denotes a set of arcs.

r_j = Net profit of visiting node *j*, which is $R_j - u_j$.

T = Maximum effective working time.

t_{dj} = Time of travelling arc (d, j).

τ_j = Service time of visiting node *j*.

$$x_{dj} = \begin{cases} 1 & \text{if arc } (d, j) \in A \text{ is traversed} \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if node } j \in V_i \text{ is visited} \\ 0 & \text{otherwise} \end{cases}$$

The pricing problem – An orienteering problem (OP)

$$Z^{i}_{price} = \max \sum_{j \in V_i} r_j y_j \ - o_i \tag{5.22}$$

Subject to:

$$\sum_{j \in V_i} x_{0j} = 2 \tag{5.23}$$

$$\sum_{(d,j) \in A} x_{dj} + \sum_{(j,k) \in A} x_{jk} = 2y_j \qquad \forall j \in V_i \tag{5.24}$$

$$\sum_{d<j} t_{dj} x_{dj} + \sum_{j \in V_i} \tau_j y_j \le T \tag{5.25}$$

$$\sum_{\substack{d \in \Lambda \\ j \in \Lambda}} x_{jd} \le |\Lambda| - 1 \qquad \forall \Lambda \subseteq V_i,\ |\Lambda| \ge 3 \tag{5.26}$$

$$x_{dj} \in \{0,1\} \qquad \forall (d,j) \in A, \tag{5.27}$$

$$y_j \in \{0,1\} \qquad \forall j \in V_i \tag{5.28}$$

In the above formulation, the objective (5.22) is to maximize the total profit generated from visiting a subset of the customers; o_i is a constant representing a dual solution corresponding to constraints (5.10). Constraint (5.23) ensures that the route must start from depot and go back to depot in the end. Constraints (5.24) indicate that, if node *j* is not included in the tour, no arcs incident to *j* are included. Otherwise, there must be one arc going into and one arc coming out of node *j*. Constraint (5.25) guarantees that the total travel and service time of the OP tour is within the time limit *T*. Constraints (5.26) are sub-tour elimination constraints, which guarantee that except a sub-tour including the depot, no other sub-tours in set V_i are allowed.

5.5.3 Procedure Involved in the Column Generation Scheme

Now, we summarize the procedure involved in the column generation scheme as follows:

Step 1: Find the initial sets $\Omega_i' \subseteq \Omega_i$ containing a feasible solution *x*.

Step 2: Solve the restricted master problem of ***P'***.

Step 3: Solve the pricing problem by heuristic.

> **Step 3.1** Initialize period count *i* = 0; *num_column* = 0;
>
> **Step 3.2** Solve the problem OP by heuristic
>
> **Step 3.3** If $Z^i{}_{price} \leq 0$, go to Step 3.4; otherwise add a column to the restricted master problem and increment *num_column*;
>
> **Step 3.4** Increment period count *i*. If *i* < *H*, go to Step 3.2.
>
> **Step 3.5** If *num_column* > 0, go to Step 2.

Step 4: Solve the problem OP by exact algorithm.

> **Step 4.1** Initialize period count *i* = 0; *num_column* = 0;
>
> **Step 4.2** Solve the pricing problem without constraints (5.26) by calling CPLEX mix-integer optimizer.
>
> **Step 4.3** If there is a sub-tour, add a sub-tour elimination constraint. Go to Step 4.2.
>
> **Step 4.4** If $Z^i{}_{price} \leq 0$, go to Step 4.5; otherwise add a column to the restricted master problem and increment *num_column*.
>
> **Step 4.5** Increment period count *i*. If *i* < *H*, go to Step 4.2.
>
> **Step 4.6** If *num_column* > 0, go to Step 2.

Step 5: Output the LP solution as an upper bound or find a feasible integer solution by calling the CPLEX mix-integer optimizer.

5.6 Computational Results and Analysis

To get the computational results, the algorithms are coded in *VISUAL C++*, and run on a 500 *MHZ* Pentium *II* Processor with 128 *MB* RAM under Microsoft Windows 98. To carry out computational experiment for the column generation scheme, we called the functions in the CPLEX optimization package into the *VC++* program.

Note that, our computational result presented below is the average performance (including solution quality and computational time) for each period, given that customers have been scheduled for service in the rolling schedule environment within 60 periods.

5.6.1 Problem Generation

For each problem instance, the location of the customers are randomly generated in the square 20*20 (km^2), according to a continuous uniform distribution. And the depot is assumed to be situated at the center. We assume that the effective working time T is 480 minutes a day. Revenue of serving a customer j, R_j is randomly generated and scaled to [0, 1]. The traveling time between customers i and j is calculated based on the Euclidean distance and the speed of the vehicle. For service time of visiting a customer j, we examine three cases in computational analysis: 10, 30 and 100 minutes, representing short, medium and long service time cases. The length of the rolling horizon is set to 7. Within each period i, the number of customers that call the company to "book" the service follows a Poisson distribution with mean arrival rate λ. In this study, λ is varied from 20 to 80 to represent the change of the problem size so that the performance of the algorithms can be evaluated. g_j, the desired period (customer *j* to be visited) that customer *j* specifies when "booking" the service, is uniformly generated within the rolling horizon. σ_j , tolerance time of visiting customer *j*, is uniformly generated within the interval [0, 2].

As described in Section 5.3 and Section 5.4, each of the heuristics HA2, HA3 and HA4 has several versions due to:

1) Different ways to construct the profit matrix p_{ij}

R1: Based on actual revenue R_j of visiting a customer *j*.

R2: Based on artificial revenue R_j of visiting customer *j* by putting more weight on assigning customer *j* to the early period of its time window.

R3: Based on artificial revenue R_j of visiting customer *j* within the desired period of service.

2) Different ways to examine the measure of desirability f_{ij}

M1: $f_{ij} = p_{ij}$

M2: $f_{ij} = p_{ij} / w_{ij}$

M3: $f_{ij} = T_i \, p_{ij} / w_{ij}$

3) Different criteria to assign customers to different periods

C1: Assign customer *j* to the earliest period of its time window.

C2: Assign customer *j* to period *i* with the largest value of p_{ij}/d_{ij}

C3: Assign customer *j* to period *i* randomly satisfying the condition that customer *j* has more chances to be visited in a period with larger value of p_{ij}/d_{ij}.

Where d_{ij} is defined as the distance between customer *j* and the center-of-gravity of the tour in period *i*.

4) Whether the algorithm is implemented based on opportunity desirability (denoted as **O**) or actual desirability (denoted as **A**).

To examine the impact of all these factors on the heuristics, we need the following notations to denote each version of the algorithms.

For HA2, we have 18 combinations altogether denoted by HA2(1) to HA2(18) as illustrated in Table 5.1, with * corresponding to the one selected in the version of the algorithm named in column 1. Similarly, for HA3, we have 6 combinations denoted by HA3(1) to HA3(6) as illustrated in Table 5.2. The 9 combinations denoted by HA4(1) to HA4(9) for heuristic HA4 are shown in Table 5.3.

Table 5.1 Denotations for heuristic HA2

HA2 versions	R1	R2	R3	M1	M2	M3	O	A
HA2(1)		*			*		*	
HA2(2)	*				*		*	
HA2(3)			*		*		*	
HA2(4)		*			*			*
HA2(5)	*				*			*
HA2(6)			*		*			*
HA2(7)		*				*	*	
HA2(8)	*					*	*	
HA2(9)			*			*	*	
HA2(10)		*				*		*
HA2(11)	*					*		*
HA2(12)			*			*		*
HA2(13)		*		*			*	
HA2(14)	*			*			*	
HA2(15)			*	*			*	
HA2(16)		*		*				*
HA2(17)	*			*				*
HA2(18)			*	*				*

Table 5.2 Denotations for heuristic HA3

HA3 versions	M1	M2	M3	O	A
HA3(1)		*		*	
HA3(2)		*			*
HA3(3)			*	*	
HA3(4)			*		*
HA3(5)	*			*	
HA3(6)	*				*

Table 5.3 Denotations for heuristic HA4

HA4 versions	R1	R2	R3	C1	C2	C3
HA4(1)		*		*		
HA4(2)	*			*		
HA4(3)			*	*		
HA4(4)		*			*	
HA4(5)	*				*	
HA4(6)			*		*	
HA4(7)		*				*
HA4(8)	*					*
HA4(9)			*			*

5.6.2 Compare the Performance of the Heuristics Against an Upper Bound

To study how the heuristics behave for the multi-period TCTSP in the rolling schedule environment, ideally, we should compare the performance of the heuristics with that of the optimal solution. However, in the rolling schedule environment, since the customer demand information is dynamically changing from one rolling horizon to the next, even the mathematical formulation of the problem is very complicated, if not impossible. Therefore, in Section 5.5, we try to find an upper bound for this problem based on set-covering type formulation and the column generation method.

In Section 5.5, we solved the LP relaxation of the set-covering model (***P’***) to optimality, which serves as an upper bound for ***Problem P***. Also, we found a feasible integer solution for ***Problem P*** based on the final basis. Results show that, the LP relaxation solution is very close to the feasible integer solution, with maximum deviation being only about 0.1%. Therefore, the LP relaxation upper bound is tight enough and we employ it as the performance measurer in the following parts.

Tables 5.4 to 5.6 illustrate the percentage deviations from the upper bound (denoted as UB) for heuristics HA1 to HA4 with the customer arrival rate and the service time being set at 20 and 100 respectively.

Table 5.4 Percentage deviations from the upper bound for HA2

Problem	UB	HA2(1)	%	HA2(2)	%	HA2(3)	%	HA2(4)	%	HA2(5)	%
1	3.53	3.40	3.87	3.39	4.12	3.08	12.77	3.28	7.05	3.43	2.92
2	3.60	3.45	4.38	3.45	4.31	3.18	11.62	3.33	7.52	3.50	2.98
3	3.64	3.52	3.17	3.52	3.24	3.23	11.06	3.38	6.93	3.54	2.73
4	3.57	3.42	4.12	3.42	4.12	3.10	13.15	3.32	7.03	3.45	3.39
5	3.62	3.47	4.13	3.47	4.13	3.20	11.55	3.40	6.06	3.49	3.46
6	3.51	3.36	4.38	3.36	4.38	3.13	10.94	3.24	7.68	3.41	2.83
7	3.59	3.41	5.04	3.42	4.67	3.17	11.66	3.30	8.03	3.46	3.66
8	3.60	3.46	3.71	3.46	3.74	3.28	8.73	3.33	7.46	3.49	3.11
9	3.58	3.42	4.35	3.42	4.35	3.13	12.49	3.35	6.49	3.49	2.61
10	3.54	3.33	5.97	3.33	5.97	3.11	12.30	3.28	7.43	3.42	3.62
11	3.60	3.43	4.73	3.44	4.40	3.17	11.89	3.31	7.99	3.50	2.81
12	3.51	3.34	4.64	3.34	4.73	3.04	13.46	3.27	6.62	3.38	3.69
13	3.54	3.38	4.59	3.39	4.41	3.13	11.74	3.28	7.33	3.45	2.72
14	3.59	3.43	4.40	3.44	4.03	3.19	11.13	3.36	6.43	3.44	4.20
15	3.55	3.42	3.75	3.42	3.75	3.14	11.49	3.29	7.36	3.44	3.06
16	3.63	3.48	4.15	3.48	4.15	3.18	12.30	3.41	5.96	3.52	2.88
17	3.56	3.36	5.51	3.36	5.51	3.11	12.70	3.31	6.97	3.42	3.77
18	3.57	3.40	4.75	3.40	4.70	3.14	12.00	3.30	7.52	3.44	3.66
19	3.59	3.44	4.19	3.44	4.19	3.17	11.85	3.27	8.95	3.46	3.64
20	3.60	3.45	3.98	3.45	3.98	3.19	11.31	3.33	7.44	3.50	2.73
Average	3.58	3.42	4.39	3.42	4.34	3.15	11.81	3.32	7.21	3.46	3.22

Table 5.4 Percentage deviations from the upper bound for HA2 (Con.)

Problem	UB	HA2(6)	%	HA2(7)	%	HA2(8)	%	HA2(9)	%	HA2(10)	%
1	3.53	3.39	4.13	3.38	4.23	3.39	4.14	3.04	13.80	3.27	7.45
2	3.60	3.41	5.35	3.43	4.78	3.43	4.73	3.13	13.04	3.34	7.18
3	3.64	3.50	3.62	3.49	3.91	3.48	4.20	3.18	12.49	3.37	7.36
4	3.57	3.38	5.24	3.42	4.20	3.42	4.20	3.11	12.82	3.33	6.87
5	3.62	3.43	5.36	3.47	4.20	3.46	4.36	3.17	12.51	3.40	6.08
6	3.51	3.34	4.99	3.36	4.29	3.36	4.29	3.10	11.79	3.28	6.69
7	3.59	3.43	4.39	3.42	4.68	3.42	4.64	3.14	12.48	3.30	8.07
8	3.60	3.46	3.86	3.45	4.08	3.46	3.97	3.21	10.88	3.33	7.40
9	3.58	3.39	5.33	3.42	4.35	3.42	4.35	3.12	12.73	3.34	6.62
10	3.54	3.36	5.25	3.33	6.13	3.33	5.93	3.07	13.28	3.28	7.43
11	3.60	3.42	4.95	3.43	4.75	3.44	4.44	3.10	13.88	3.32	7.63
12	3.51	3.29	6.21	3.35	4.51	3.35	4.63	3.02	14.00	3.28	6.53
13	3.54	3.33	5.98	3.38	4.60	3.38	4.60	3.11	12.21	3.29	7.13
14	3.59	3.41	4.82	3.45	3.82	3.44	3.95	3.15	12.09	3.34	6.83
15	3.55	3.38	4.68	3.42	3.64	3.42	3.64	3.09	12.83	3.26	8.20
16	3.63	3.45	4.75	3.47	4.34	3.47	4.33	3.16	12.78	3.42	5.79
17	3.56	3.41	4.18	3.36	5.54	3.36	5.70	3.12	12.33	3.33	6.34
18	3.57	3.37	5.40	3.40	4.74	3.40	4.69	3.08	13.73	3.27	8.24
19	3.59	3.42	4.86	3.43	4.56	3.43	4.47	3.16	12.05	3.28	8.77
20	3.60	3.42	4.94	3.45	3.97	3.45	3.97	3.19	11.44	3.36	6.46
Average	3.58	3.40	4.91	3.40	4.47	3.40	4.46	3.12	12.66	3.32	7.15

Table 5.4 Percentage deviations from the upper bound for HA2 (Con.)

Problem	UB	HA2(11)	%	HA2(12)	%	HA2(13)	%	HA2(14)	%
1	3.53	3.44	2.73	3.38	4.20	3.40	3.66	3.39	3.92
2	3.60	3.52	2.34	3.42	5.19	3.47	3.58	3.48	3.49
3	3.64	3.55	2.33	3.49	3.99	3.55	2.30	3.55	2.30
4	3.57	3.48	2.65	3.40	4.66	3.45	3.36	3.45	3.36
5	3.62	3.53	2.53	3.43	5.14	3.50	3.38	3.50	3.25
6	3.51	3.42	2.56	3.35	4.55	3.40	3.31	3.40	3.31
7	3.59	3.47	3.27	3.42	4.58	3.46	3.66	3.46	3.64
8	3.60	3.51	2.31	3.46	3.88	3.50	2.86	3.50	2.86
9	3.58	3.49	2.58	3.39	5.41	3.45	3.56	3.45	3.56
10	3.54	3.44	2.83	3.35	5.45	3.36	5.10	3.37	4.99
11	3.60	3.52	2.03	3.43	4.74	3.44	4.34	3.45	4.01
12	3.51	3.41	2.66	3.30	5.98	3.37	3.96	3.37	4.05
13	3.54	3.45	2.58	3.34	5.58	3.41	3.87	3.41	3.71
14	3.59	3.50	2.43	3.42	4.62	3.46	3.49	3.45	3.69
15	3.55	3.47	2.25	3.38	4.64	3.44	2.96	3.44	2.96
16	3.63	3.55	2.00	3.45	4.87	3.52	2.83	3.52	2.83
17	3.56	3.46	2.76	3.39	4.81	3.42	3.87	3.42	3.99
18	3.57	3.48	2.52	3.36	5.68	3.43	3.85	3.42	4.01
19	3.59	3.50	2.58	3.42	4.88	3.48	3.17	3.48	3.05
20	3.60	3.51	2.38	3.42	4.92	3.49	2.91	3.49	2.91
Average	3.58	3.47	2.52	3.37	4.89	3.45	3.50	3.45	3.49

Table 5.4 Percentage deviations from the upper bound for HA2 (Con.)

Problem	UB	HA2(15)	%	HA2(16)	%	HA2(17)	%	HA2(18)	%
1	3.53	3.06	13.24	3.27	7.54	3.44	2.59	3.40	3.86
2	3.60	3.19	11.39	3.31	8.17	3.52	2.29	3.44	4.41
3	3.64	3.24	11.01	3.36	7.58	3.58	1.58	3.49	3.91
4	3.57	3.10	13.24	3.30	7.69	3.49	2.13	3.40	4.82
5	3.62	3.20	11.44	3.42	5.43	3.56	1.64	3.45	4.69
6	3.51	3.11	11.51	3.26	7.08	3.44	2.10	3.37	4.10
7	3.59	3.17	11.59	3.33	7.10	3.48	2.97	3.44	3.97
8	3.60	3.28	8.97	3.32	7.86	3.51	2.31	3.46	3.93
9	3.58	3.13	12.54	3.35	6.34	3.52	1.73	3.44	3.84
10	3.54	3.10	12.59	3.30	6.96	3.43	3.31	3.35	5.35
11	3.60	3.18	11.56	3.34	7.15	3.53	1.72	3.44	4.41
12	3.51	3.04	13.42	3.27	6.65	3.41	2.87	3.33	4.97
13	3.54	3.14	11.46	3.29	7.19	3.45	2.64	3.34	5.67
14	3.59	3.18	11.23	3.37	6.13	3.51	2.00	3.44	3.95
15	3.55	3.14	11.58	3.25	8.33	3.48	1.96	3.39	4.53
16	3.63	3.21	11.61	3.40	6.37	3.54	2.26	3.46	4.50
17	3.56	3.12	12.39	3.32	6.78	3.47	2.42	3.40	4.35
18	3.57	3.13	12.30	3.29	7.76	3.48	2.41	3.39	5.04
19	3.59	3.19	11.27	3.30	8.08	3.52	1.93	3.43	4.46
20	3.60	3.20	10.94	3.34	7.26	3.53	1.88	3.43	4.52
Average	3.58	3.15	11.76	3.32	7.17	3.50	2.24	3.42	4.46

For HA2, HA3 and HA4, Tables 5.4 to 5.6 show that, the factors considered in the algorithms (R1, R2, R3; M1, M2, M3; O, A; and C1, C2, C3) all exert influence on the solution quality of the algorithms to some extent.

One finding from Table 5.4 is that, constructing the profit matrix p_{ij} based on actual revenue (R1) yields better solution than the other two methods (R2 and R3). Moreover, whether employing opportunity desirability (O) or actual desirability (A) can provide better solution depends on how to construct the profit matrix p_{ij}: if it is constructed by R1 or R3, selecting the customers based on actual desirability (A) performs better; otherwise, one should select the customers based on opportunity desirability (O). For the measure of desirability f_{ij}, if the algorithm is implemented based on opportunity desirability (O), f_{ij} defined by M1 provides the best solution followed by M2 and M3; otherwise, f_{ij} defined by M1 provides the best solution followed by M3 and M2. Another observation is that, the average percentage deviation is quite different from one version of the algorithm to another, which can be as small as 2.24% and as large as 12.66%. Among all 18 combinations, the smallest average percentage deviation from the upper bound, 2.24%, is generated by HA2(17), which is a combination of M1AR1, that is: construct the profit matrix p_{ij} with actual revenue (R1); select the customers based on actual desirability (A) and define the measure of desirability by $f_{ij} = p_{ij}$.

Table 5.5 Percentage deviations from the upper bound for HA3

Problem	UB	HA3(1)	%	HA3(2)	%	HA3(3)	%	HA3(4)	%	HA3(5)	%	HA3(6)	%
1	3.53	3.44	2.48	3.40	3.77	3.45	2.40	3.43	3.00	3.46	1.97	3.45	2.21
2	3.60	3.50	2.88	3.49	3.06	3.49	3.24	3.51	2.70	3.53	2.14	3.50	2.77
3	3.64	3.54	2.58	3.53	2.86	3.55	2.31	3.54	2.62	3.58	1.67	3.57	1.88
4	3.57	3.46	3.12	3.45	3.35	3.47	2.85	3.48	2.57	3.51	1.84	3.50	2.07
5	3.62	3.54	2.28	3.48	3.82	3.53	2.34	3.53	2.43	3.55	1.79	3.56	1.59
6	3.51	3.43	2.47	3.40	3.16	3.43	2.40	3.42	2.61	3.46	1.41	3.44	1.97
7	3.59	3.47	3.20	3.48	2.91	3.46	3.52	3.45	3.71	3.49	2.78	3.49	2.57
8	3.60	3.49	3.11	3.48	3.39	3.51	2.49	3.51	2.58	3.54	1.58	3.53	1.81
9	3.58	3.49	2.52	3.47	3.10	3.47	3.07	3.49	2.49	3.51	1.97	3.50	2.20
10	3.54	3.44	2.96	3.42	3.52	3.45	2.70	3.43	3.09	3.46	2.37	3.45	2.70
11	3.60	3.51	2.36	3.49	2.94	3.52	2.26	3.51	2.37	3.54	1.44	3.51	2.40
12	3.51	3.42	2.58	3.40	3.12	3.41	2.78	3.40	3.04	3.41	2.81	3.42	2.55
13	3.54	3.43	3.18	3.42	3.43	3.43	3.10	3.46	2.45	3.46	2.31	3.47	1.91
14	3.59	3.50	2.38	3.46	3.57	3.50	2.37	3.50	2.46	3.52	1.90	3.50	2.37
15	3.55	3.44	3.02	3.41	3.91	3.44	3.21	3.45	2.67	3.48	2.01	3.46	2.40
16	3.63	3.54	2.41	3.50	3.36	3.55	2.05	3.54	2.45	3.57	1.66	3.56	1.86
17	3.56	3.46	2.79	3.43	3.51	3.45	3.08	3.47	2.55	3.49	1.86	3.46	2.82
18	3.57	3.47	2.82	3.42	4.17	3.46	3.09	3.48	2.37	3.49	2.04	3.49	2.29
19	3.59	3.48	3.22	3.47	3.31	3.50	2.66	3.48	3.21	3.52	1.99	3.53	1.64
20	3.60	3.49	2.94	3.47	3.40	3.50	2.80	3.48	3.15	3.54	1.50	3.53	1.80
Average	3.58	3.48	2.76	3.45	3.38	3.48	2.74	3.48	2.73	3.51	1.95	3.50	2.19

For HA3, Table 5.5 clearly shows that, if measure of desirability f_{ij} is defined by M1 or M2, algorithms implemented based on opportunity desirability (O) are superior to those based on actual desirability (A); while for f_{ij} defined by M3, the average percentage deviations from the upper bound for the two alternatives are very similar to each other. The smallest deviation, 1.95%, is obtained from HA3(5), with measure of desirability f_{ij} defined by M1 and implement the algorithm based on opportunity desirability (O). The overall performance of HA3 is quite good, with average percentage deviations from the upper bound for all the versions being within 3.5%.

Table 5.6 Percentage deviations from the upper bound for HA4 and HA1

Problem	UB	HA4(1)	%	HA4(2)	%	HA4(3)	%	HA4(4)	%	HA4(5)	%
1	3.53	3.29	6.81	3.50	0.84	3.47	1.87	3.28	7.22	3.46	2.16
2	3.60	3.35	6.98	3.55	1.35	3.50	2.90	3.32	7.82	3.52	2.21
3	3.64	3.37	7.23	3.60	1.10	3.55	2.36	3.37	7.41	3.60	1.11
4	3.57	3.31	7.22	3.54	0.96	3.48	2.55	3.31	7.40	3.49	2.21
5	3.62	3.42	5.52	3.59	0.73	3.52	2.68	3.42	5.48	3.56	1.52
6	3.51	3.26	7.10	3.49	0.69	3.44	2.12	3.26	7.30	3.46	1.50
7	3.59	3.33	7.04	3.56	0.80	3.52	1.93	3.33	7.28	3.51	2.05
8	3.60	3.33	7.47	3.56	1.02	3.54	1.60	3.33	7.55	3.54	1.74
9	3.58	3.36	6.10	3.55	0.79	3.50	2.25	3.35	6.31	3.51	1.85
10	3.54	3.31	6.55	3.52	0.72	3.42	3.40	3.32	6.37	3.46	2.37
11	3.60	3.37	6.30	3.58	0.57	3.50	2.61	3.36	6.61	3.54	1.70
12	3.51	3.28	6.49	3.47	1.12	3.41	2.85	3.27	6.75	3.43	2.20
13	3.54	3.28	7.29	3.49	1.46	3.44	2.79	3.27	7.65	3.45	2.68
14	3.59	3.39	5.48	3.55	0.93	3.53	1.62	3.37	5.98	3.47	3.11
15	3.55	3.28	7.55	3.50	1.44	3.45	2.93	3.27	7.92	3.46	2.47
16	3.63	3.42	5.76	3.59	1.14	3.54	2.42	3.42	5.74	3.57	1.45
17	3.56	3.33	6.48	3.52	1.09	3.46	2.64	3.31	7.01	3.47	2.55
18	3.57	3.29	7.63	3.52	1.24	3.47	2.66	3.28	8.16	3.48	2.34
19	3.59	3.34	7.12	3.56	0.91	3.50	2.52	3.32	7.62	3.51	2.29
20	3.60	3.33	7.47	3.55	1.39	3.51	2.50	3.32	7.72	3.54	1.72
Average	3.58	3.33	6.78	3.54	1.02	3.49	2.46	3.32	7.07	3.50	2.06

Table 5.6 Percentage deviations from the upper bound for HA4 and HA1 (Con.)

Problem	UB	HA4(6)	%	HA4(7)	%	HA4(8)	%	HA4(9)	%	HA1	%
1	3.53	3.40	3.83	3.37	4.70	3.37	4.70	2.95	16.56	3.47	1.84
2	3.60	3.42	5.01	3.44	4.48	3.44	4.48	3.07	14.71	3.55	1.37
3	3.64	3.48	4.17	3.49	4.10	3.49	4.10	3.13	14.04	3.61	0.76
4	3.57	3.38	5.41	3.44	3.66	3.44	3.66	3.08	13.82	3.52	1.39
5	3.62	3.47	4.18	3.47	4.18	3.47	4.18	3.08	14.77	3.58	1.13
6	3.51	3.36	4.31	3.38	3.89	3.38	3.89	3.05	13.14	3.47	1.11
7	3.59	3.41	4.88	3.42	4.79	3.42	4.79	3.05	14.85	3.52	1.82
8	3.60	3.45	4.01	3.43	4.57	3.43	4.57	3.16	12.14	3.56	1.00
9	3.58	3.44	3.95	3.44	3.96	3.44	3.96	3.11	13.28	3.54	1.24
10	3.54	3.34	5.80	3.33	6.07	3.33	6.07	3.03	14.56	3.48	1.67
11	3.60	3.44	4.47	3.44	4.48	3.44	4.48	3.05	15.18	3.56	0.97
12	3.51	3.32	5.31	3.33	5.18	3.33	5.18	2.93	16.44	3.44	1.97
13	3.54	3.33	5.96	3.39	4.26	3.39	4.26	3.07	13.36	3.47	2.10
14	3.59	3.43	4.48	3.43	4.35	3.43	4.35	3.09	13.86	3.55	0.94
15	3.55	3.37	4.92	3.43	3.49	3.43	3.49	3.03	14.71	3.49	1.60
16	3.63	3.46	4.49	3.49	3.70	3.49	3.70	3.11	14.17	3.59	1.03
17	3.56	3.41	4.25	3.37	5.16	3.37	5.16	3.05	14.36	3.50	1.68
18	3.57	3.36	5.88	3.39	5.09	3.39	5.09	2.99	16.14	3.52	1.40
19	3.59	3.41	5.22	3.45	4.09	3.45	4.09	3.10	13.77	3.54	1.50
20	3.60	3.43	4.62	3.45	4.14	3.45	4.14	3.11	13.48	3.54	1.64
Average	3.58	3.41	4.76	3.42	4.42	3.42	4.42	3.06	14.37	3.53	1.41

For HA4, assign the customers to the periods based on criterion C1 is better than the other two criteria: C2 and C3. Once customers are assigned to a period, selecting the customers based on p_{ij}/d_{ij} with $p_{ij} = R_j$ (constructing the profit matrix based on the actual revenue) provides the best solution. The smallest average percentage deviation (1.02%), is obtained from HA4(2). Unlike HA3, solution quality of different versions of HA4 heavily depends on how to assign the customers (C1, C2 and C3) and how to construct the profit matrix p_{ij} for selecting the customers within one period.

For HA1, the average percentage deviation from the upper bound is 1.41%. It is better than the best among both HA2 (2.24%) and HA3 (1.95%), only inferior to the best of HA4 (1.02%).

The average computational time taken to get the best solution for each of the heuristics is illustrated in Table 5.7, which indicates that HA1 is the least time consuming heuristic, followed by HA2, HA4 and HA3.

Table 5.7 Computational time taken to get the best solution for each heuristic

Customer Arrival Rate	Computational time (ms)			
	HA1	HA2	HA3	HA4
20	11	24	57	41

5.6.3 Performance Comparison Among the Heuristics

For problems with high customer arrival rate and short service times, it is very difficult to get the upper bound in terms of computational time. Therefore, the performance of the heuristics is compared among themselves to determine which one is more preferable in a certain scenario. As we discussed above, each heuristic has several versions due to different combinations of the factors involved in it; totally, we have 34 versions of the heuristics. For each problem instance, the heuristic with the maximum profit realized is identified and regarded as the performance measurer; then each heuristic is examined by calculating the percentage deviation from the maximum profit. The following computational results are the averages of the results obtained from 30 randomly generated problem instances.

5.6.3.1 Analysis of the Factors Affecting the Performance of the Heuristics

Effect of different profit matrix, measure of desirability and opportunity or actual desirability in HA2

Tables 5.8 to 5.10 illustrate the average percentage deviations from the maximum for HA2 when the service time is 10, 30 and 100 minutes respectively. The minimum average percentage deviation among all 18 combinations for each case is shown in bold and italic font.

Table 5.8 Average percentage deviations from the maximum for HA2

(Service time = 10 minutes)

λ	HA2(1)	HA2(2)	HA2(3)	HA2(4)	HA2(5)	HA2(6)	HA2(7)	HA2(8)	HA2(9)
20	0.45	1.37	1.32	0.86	2.07	1.64	0.90	2.11	1.62
30	0.29	1.28	1.18	2.15	3.83	2.15	1.35	2.54	1.66
40	3.12	3.39	3.76	3.00	4.26	2.31	3.27	3.60	3.80
50	3.38	3.65	4.60	2.58	3.14	1.80	3.48	3.72	4.62
60	3.82	4.05	5.55	2.35	2.52	1.75	3.89	4.14	5.53
70	4.44	4.61	6.42	2.18	1.70	1.73	4.45	4.61	6.39
80	4.82	4.93	7.11	2.05	1.46	2.01	4.77	4.89	7.06

Table 5.8 Average percentage deviations from the maximum for HA2 (Con.)

(Service time = 10 minutes)

λ	HA2(10)	HA2(11)	HA2(12)	HA2(13)	HA2(14)	HA2(15)	HA2(16)	HA2(17)	HA2(18)
20	0.71	1.69	1.51	*0.30*	3.15	1.33	0.32	3.15	1.34
30	0.74	1.64	1.37	*0.20*	3.23	1.19	0.54	3.27	1.43
40	*0.60*	1.20	1.01	3.15	3.93	3.90	1.24	3.37	1.41
50	0.79	*0.58*	0.78	3.39	4.06	4.84	1.66	3.26	1.47
60	0.97	*0.64*	1.21	3.76	4.36	6.09	2.77	3.67	2.22
70	0.96	*0.38*	1.36	4.10	4.65	7.34	3.46	3.92	2.70
80	0.91	*0.25*	1.79	4.51	4.93	8.64	4.27	4.29	3.10

Table 5.9 Average percentage deviations from the maximum for HA2

(Service time = 30 minutes)

λ	HA2(1)	HA2(2)	HA2(3)	HA2(4)	HA2(5)	HA2(6)	HA2(7)	HA2(8)	HA2(9)
20	3.80	4.13	5.56	3.16	3.23	2.06	3.84	4.19	5.53
30	3.84	3.96	7.65	3.44	1.86	2.80	3.83	3.97	7.67
40	3.80	3.85	9.73	3.67	1.17	2.93	3.57	3.67	9.68
50	4.18	4.23	11.64	4.46	0.92	3.20	3.96	4.02	11.75
60	4.74	4.77	13.35	5.18	0.82	3.67	4.61	4.67	13.75
70	5.07	5.08	14.48	5.88	0.90	3.87	4.89	4.91	15.06
80	5.41	5.38	14.99	6.44	0.83	4.30	5.34	5.26	15.74

Table 5.9 Average percentage deviations from the maximum for HA2 (Con.)

(Service time = 30 minutes)

λ	HA2(10)	HA2(11)	HA2(12)	HA2(13)	HA2(14)	HA2(15)	HA2(16)	HA2(17)	HA2(18)
20	1.25	*0.50*	1.50	4.13	4.87	5.95	3.01	3.28	2.52
30	2.27	*0.34*	2.56	4.54	4.97	8.65	4.74	3.62	3.56
40	2.95	*0.32*	2.87	4.84	5.15	11.77	6.06	3.88	3.73
50	3.83	*0.15*	3.15	5.37	5.43	14.26	7.69	4.35	4.15
60	4.92	*0.24*	3.63	6.05	6.04	16.65	9.36	4.95	4.66
70	5.48	*0.24*	3.82	6.64	6.63	17.62	10.25	5.44	5.01
80	6.31	*0.14*	4.28	7.10	7.11	17.95	11.00	6.04	5.53

Table 5.10 Average percentage deviations from the maximum for HA2

(Service time = 100 minutes)

λ	HA2(1)	HA2(2)	HA2(3)	HA2(4)	HA2(5)	HA2(6)	HA2(7)	HA2(8)	HA2(9)
30	3.04	3.04	9.91	6.44	2.28	3.65	3.10	3.09	11.36
40	3.19	3.17	8.56	6.11	2.39	3.51	3.31	3.31	9.96
50	3.04	3.02	7.16	5.86	2.37	3.40	3.10	3.08	8.59
60	3.10	3.08	6.55	5.59	2.50	3.36	3.12	3.10	7.93
70	3.10	3.07	6.09	5.32	2.62	3.37	3.19	3.19	7.17
80	3.20	3.19	5.74	4.97	2.62	3.38	3.22	3.20	6.82

Table 5.10 Average percentage deviations from the maximum for HA2 (Con.)

(Service time = 100 minutes)

λ	HA2(10)	HA2(11)	HA2(12)	HA2(13)	HA2(14)	HA2(15)	HA2(16)	HA2(17)	HA2(18)
30	6.41	1.65	3.66	1.77	1.77	9.81	6.26	*0.79*	2.78
40	6.09	2.03	3.50	1.51	1.49	8.01	5.59	*0.61*	2.03
50	5.83	2.19	3.40	1.07	1.05	6.52	5.05	*0.43*	1.67
60	5.57	2.37	3.37	0.95	0.95	5.74	4.53	*0.39*	1.48
70	5.29	2.53	3.36	0.88	0.89	4.91	3.94	*0.45*	1.26
80	4.96	2.62	3.38	0.76	0.76	4.48	3.40	*0.33*	1.11

Results from Tables 5.8 to 5.10 indicate that, there is no dominant combination always superior to the others. Specifically, when the service time is 10 minutes and when the customer arrival rate is low (λ < 40), HA2(13) with the combination **M1OR2** performs the best; however, when λ = 40, HA2(10) with the combination **M3AR2** yields the best solution. In case of λ > 40 and the case when the service time is medium, HA2(11) with the combination **M3AR1** gives the best result. Finally, when the service time is long (100 minutes), HA2(17) with the combination **M1AR1** is the best alternative. The combination with the best performance in each scenario is summarized in Table 5.11.

Table 5.11 Combinations with the best performance for HA2, HA3 and HA4

Heuristic	Customer Arrival Rate λ	Service time (minutes)		
		10	30	100
HA2	$\lambda < 40$	M1OR2	M3AR1	M1AR1
	$\lambda = 40$	M3AR2	M3AR1	M1AR1
	$\lambda > 40$	M3AR1	M3AR1	M1AR1
HA3	$\lambda < 50$	M1A	M3A	M1O
	$\lambda \geq 50$	M3A	M3A	M1O
HA4	$\lambda \leq 40$	C1R3	C2R1	C1R1
	$\lambda > 50$	C2R1	C2R1	C1R1

Computational results show that, all three factors considered in HA2 have some impacts on the performance of the heuristics, and the factors are interdependent. Figure 5.1 illustrates the effect of different measure of desirability (**M1**, **M2** and **M3**) on HA2 when actual desirability (**A**) is applied and when profit matrix is constructed by **R1**, with service time being set at 100 minutes. In this case, it is clear that, desirability defined by $f_{ij} = p_{ij}$ (**M1**) gives the best solution; while $f_{ij} = p_{ij} / w_{ij}$ (**M2**) gives the worst solution.

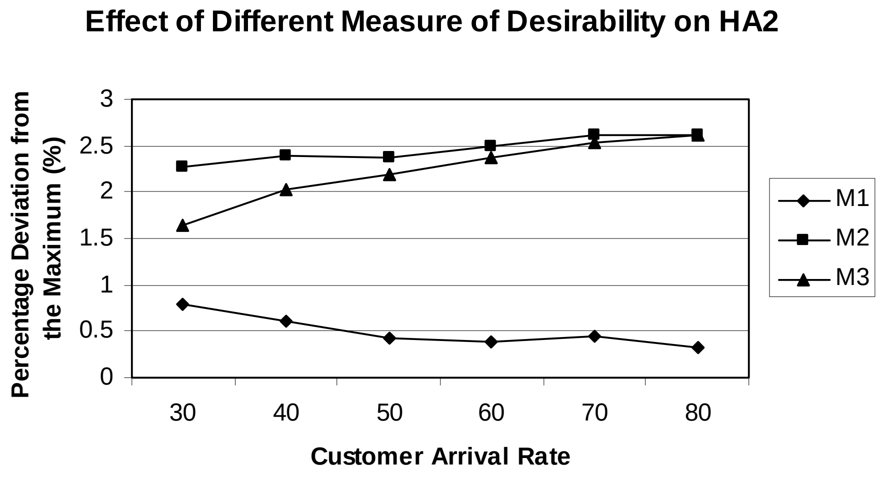


Figure 5.1 Effect of Different Measure of Desirability on Heuristic HA2

The effect of different profit matrix (**R1**, **R2** and **R3**) on HA2 is shown in Figure 5.2 when actual desirability (**A**) is applied and when measure of desirability is fixed at **M3**, with the service time being set at 10 minutes. In this case, when the customer arrival rate is less than 50, we should construct the profit matrix according to **R2**, that is, putting more weight on assigning customer *j* to the early period of its time window. Otherwise, profit matrix should be constructed based on R1, the actual revenue.

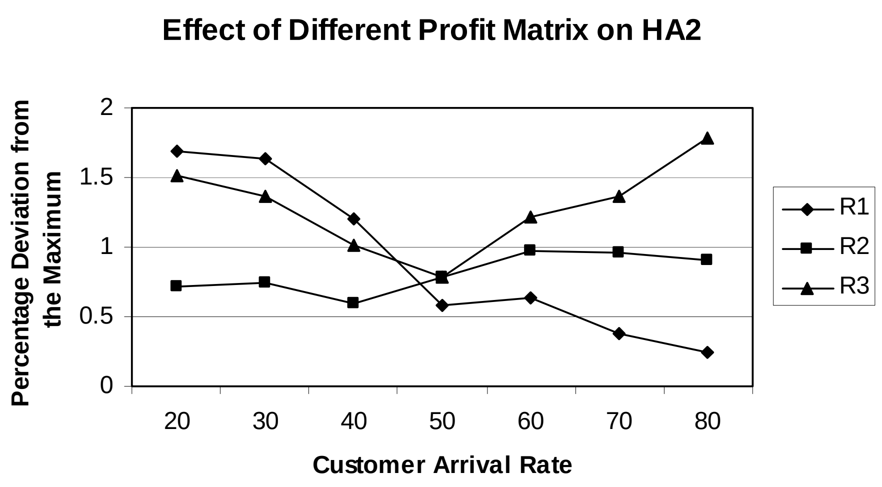


Figure 5.2 Effect of Different Profit Matrix on HA2

Effect of applying different measure of desirability and opportunity or actual desirability in H3

Tables 5.12 to 5.14 illustrate the average percentage deviation from the maximum for HA3 when the service time is 10, 30 and 100 minutes respectively. Similarly, the minimum average percentage deviation from the maximum among the 6 combinations is shown in bold and italic font.

Table 5.12 Average percentage deviations from the maximum for HA3 and HA1

(Service time = 10 minutes)

λ	HA3(1)	HA3(2)	HA3(3)	HA3(4)	HA3(5)	HA3(6)	HA1
20	2.15	2.19	2.20	1.66	1.88	*0.10*	1.72
30	2.52	3.73	2.28	1.61	2.41	*0.20*	1.83
40	2.62	4.17	2.09	1.23	2.42	*0.72*	0.96
50	2.05	3.15	1.64	*0.72*	1.98	0.77	0.17
60	1.75	2.30	1.40	*0.62*	1.87	1.09	0.17
70	1.59	1.62	1.12	*0.37*	1.89	1.18	0.26
80	1.16	1.33	1.08	*0.33*	2.10	1.36	0.35

Table 5.13 Average percentage deviations from the maximum for HA3 and HA1

(Service time = 30 minutes)

λ	HA3(1)	HA3(2)	HA3(3)	HA3(4)	HA3(5)	HA3(6)	HA1
20	1.78	3.34	1.32	*0.52*	1.72	1.52	0.43
30	1.17	1.87	0.78	*0.34*	1.77	1.73	0.62
40	1.03	0.99	0.84	*0.43*	1.95	1.72	0.92
50	0.79	0.76	0.73	*0.21*	2.52	2.27	1.43
60	0.86	0.73	0.75	*0.36*	3.03	3.02	2.14
70	0.71	0.73	0.55	*0.23*	3.56	3.63	2.73
80	0.69	0.74	0.58	*0.27*	4.20	4.09	3.23

Table 5.14 Average percentage deviations from the maximum for HA3 and HA1

(Service time = 100 minutes)

λ	HA3(1)	HA3(2)	HA3(3)	HA3(4)	HA3(5)	HA3(6)	HA1
30	2.00	2.40	1.94	1.98	*0.61*	0.80	0.25
40	2.29	2.52	2.23	2.27	*0.53*	0.68	0.20
50	2.48	2.45	2.38	2.36	*0.44*	0.55	0.10
60	2.60	2.70	2.61	2.63	*0.50*	0.55	0.13
70	2.72	2.81	2.74	2.77	*0.44*	0.55	0.16
80	2.82	2.88	2.84	2.83	*0.41*	0.50	0.14

The combination with the best performance in each scenario is also summarized in Table 5.11. Specifically, when the service time is short and medium, actual desirability should be selected; otherwise, one should choose opportunity desirability. In terms of measure of desirability f_{ij}, when the service time is short with arrival rate $\lambda < 50$ and when the service time is long, **M1** is the best; **M3** should be applied when the service time is short with arrival rate $\lambda \geq 50$ and when the service time is medium.

For HA3, the overall performance is quite good with largest percentage deviation (HA3(5) when service time is 30 minutes) from the maximum being within 4.2%. The impact of actual or opportunity desirability (**A/O**) is marginal except for the case when service time is set at 10 minutes. Since the effect of different measure of desirability is more important, we illustrate in Figure 5.3 of this effect in the case when service time is 30 minutes and when we assign based on actual desirability. In this case, measure of desirability defined by $f_{ij} = T_i \, p_{ij} / w_{ij}$ (**M3**) gives the best solution.

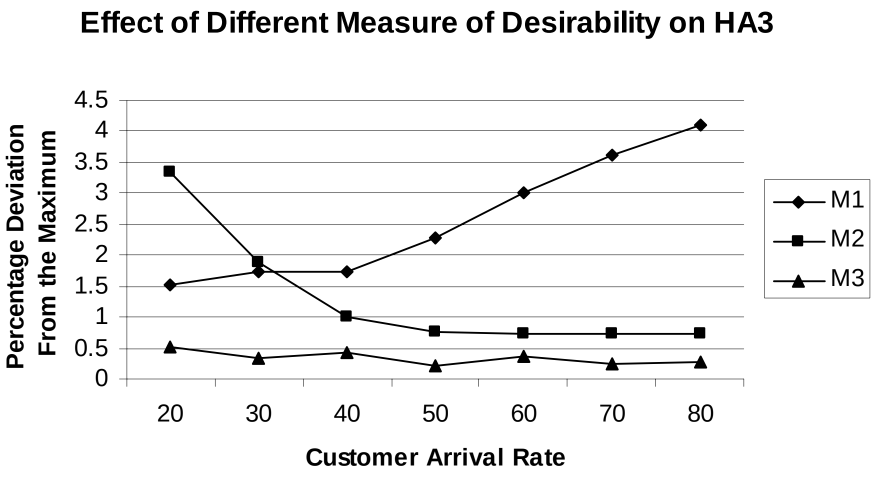


Figure 5.3 Effect of Different Measure of Desirability on Heuristic HA3

Effect of artificial revenue and different assigning criteria in H4

Tables 5.15 to 5.17 illustrate the average percentage deviations from the maximum for HA4 when service time is 10, 30 and 100 minutes respectively. The minimum average percentage deviation from the maximum among the 9 combinations is shown in bold and italic font. Specifically, when service time is short, and customer arrival rate $\lambda < 50$, HA4(3) gives the best result, where profit matrix is constructed by **R3**, and customers are assigned by **C1**; when the service time is short and customer arrival rate $\lambda \geq 50$, and when service time is medium, HA4(5) yields the best result with profit matrix constructed by **R1**, and customers assigned by **C2**; when service time is long, HA4(2) provides the best solution, again profit matrix should be constructed by **R1**, but customers should be assigned based on **C1** in this case. The best combination in different scenarios for HA4 can be found in Table 5.11.

Table 5.15 Average percentage deviations from the maximum for HA4

(Service time = 10 minutes)

λ	HA4(1)	HA4(2)	HA4(3)	HA4(4)	HA4(5)	HA4(6)	HA4(7)	HA4(8)	HA4(9)
20	0.00	0.00	*0.00*	0.14	1.62	1.38	3.15	3.15	3.15
30	0.25	0.13	*0.02*	0.36	1.66	1.44	3.56	3.56	3.60
40	0.81	0.25	*0.00*	0.91	1.14	1.35	4.34	4.34	4.79
50	1.27	0.33	0.34	1.32	*0.14*	1.50	4.27	4.27	5.48
60	2.32	1.01	1.17	2.23	*0.11*	2.21	4.53	4.54	6.68
70	3.30	1.48	1.93	3.08	*0.15*	2.69	4.81	4.81	7.94
80	4.26	2.02	2.48	3.92	*0.25*	3.01	5.10	5.09	9.17

Table 5.16 Average percentage deviations from the maximum for HA4

(Service time = 30 minutes)

λ	HA4(1)	HA4(2)	HA4(3)	HA4(4)	HA4(5)	HA4(6)	HA4(7)	HA4(8)	HA4(9)
20	2.05	0.46	0.20	2.29	*0.46*	2.53	5.34	5.34	6.97
30	4.32	1.57	1.99	4.17	*0.17*	3.60	5.25	5.24	9.48
40	6.24	2.23	2.90	6.14	*0.30*	3.70	5.43	5.43	12.37
50	8.15	3.17	3.88	7.90	*0.63*	4.17	5.71	5.72	14.99
60	9.71	4.05	5.00	9.48	*1.28*	4.66	6.27	6.27	17.70
70	10.88	4.62	5.52	10.57	*1.78*	4.96	6.84	6.84	19.22
80	11.46	5.36	6.08	11.23	*2.28*	5.36	7.13	7.13	19.94

Table 5.17 Average percentage deviations from the maximum for HA4

(Service time = 100 minutes)

λ	HA4(1)	HA4(2)	HA4(3)	HA4(4)	HA4(5)	HA4(6)	HA4(7)	HA4(8)	HA4(9)
30	6.27	*0.05*	1.46	6.32	0.58	2.82	2.24	2.24	13.98
40	5.57	*0.10*	1.09	5.53	0.58	2.03	2.08	2.08	12.79
50	5.02	*0.13*	0.84	5.02	0.26	1.66	1.39	1.39	10.79
60	4.49	*0.08*	0.86	4.51	0.30	1.49	1.22	1.22	9.65
70	3.93	*0.13*	0.71	3.93	0.33	1.23	1.26	1.26	8.42
80	3.38	*0.15*	0.73	3.38	0.27	1.11	0.95	0.95	7.54

Computational results reveal that, except for the case when service time is 10 minutes, profit matrix constructed by **R1**, actual revenue, always obtains the best solution. Figure 5.4 displays this effect when customers are assigned based on **C2** and service time is set at 30 minutes.

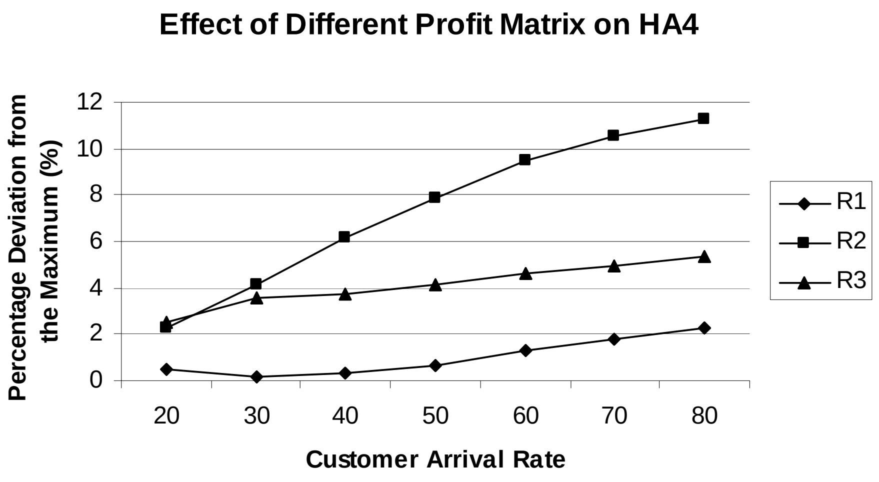


Figure 5.4 Effect of Different Profit Matrix on Heuristic HA4

For the effect of different assigning criteria, computational results show that, when the service time is short with arrival rate $\lambda < 50$ and when the service time is long, we should assign the customers according to criterion **C1**. Otherwise, when the service time is short with arrival rate $\lambda \geq 50$ and when the service time is medium, we should assign the customers according to criterion **C2**. An example of the above effect for the case of medium service time is illustrated in Figure 5.5, which shows that criterion **C2** produces the best result.

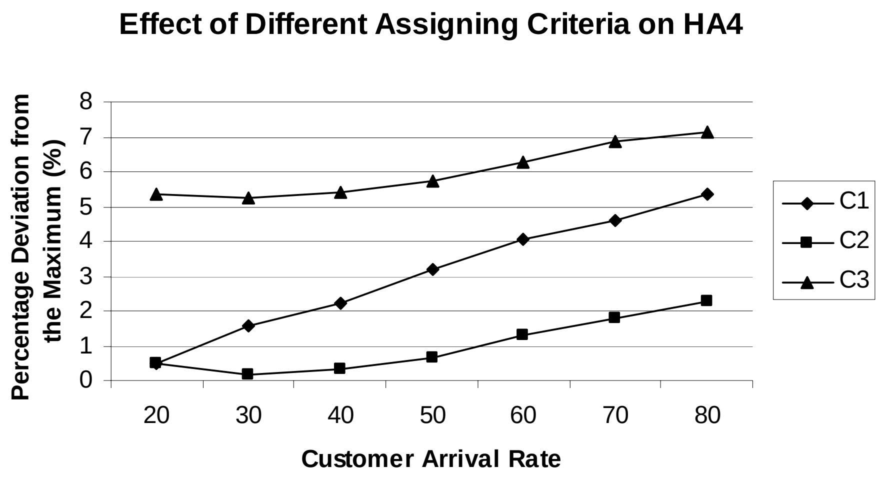


Figure 5.5 Effect of Different Assigning Criteria on Heuristic HA4

Some observations

From the above analysis and data in Table 5.11, the following observations can be obtained. We classify all the scenarios considered above into three types. Type I: when customer arrival rate is low and service time is short, most customers can be included in the solution. Type II: when the service time is very long, only a few customers can be included in the solution. Type III: when the service time is short and customer arrival rate is high, or when service time is medium, the number of customers can be included in the solution is neither very large nor very small. For the measure of desirability in HA2 and HA3, in type I and II, it should be defined by $f_{ij} = p_{ij}$, i.e., only consider the revenue of assigning customer *j* in period *i*. However, in type III, it should be defined by $f_{ij} = T_i p_{ij} / w_{ij}$, i.e., we should not only consider the revenue of the customer, but also the time required to visit it, and the remaining working time at the moment. For HA4, customers should be assigned according to criterion **C1** for type I and II and according to criterion **C2** in type III. For profit matrix in HA2 and HA4, it should be constructed based on actually revenue (**R1**) for type II and III; for type I, in HA2, it should be constructed by **R2**, putting more weight on the earlier period the customer requests a service; in HA4, it should be constructed by **R3**, putting more weight on the period the customer requests the service. For the opportunity or actual desirability, in HA2, opportunity desirability should be applied in type I; otherwise, actual desirability should be used. In HA3, for type II, opportunity desirability should be applied; otherwise, actual desirability should be applied.

Finally, the heuristic yields the best solution in each scenario is summarized in Table 5.18.

Table 5.18 The heuristic yields the best solution in different scenarios

Customer Arrival Rate λ	Service time (minutes)		
	10	30	100
20	HA4(3)	HA4(5)	HA4(2)
30	HA4(3)	HA4(5)	HA4(2)
40	HA4(3)	HA4(5)	HA4(2)
50	HA4(5)	HA2(11)	HA1
60	HA4(5)	HA2(11)	HA4(2)
70	HA4(5)	HA3(4)	HA4(2)
80	HA4(5) or HA2(11)	HA2(11)	HA1

5.6.3.2 Comparison of the Heuristic Performances

We make a comparison of the performance of the heuristics HA1, HA2, HA3 and HA4 in this section. We compare the performance of the heuristics in several scenarios: the service time is short, the service time is medium, and the service time is long. Since each heuristic (HA2, HA3 and HA4) has several versions due to the factors discussed in Section 5.6.3.1, we select the ones with the best solution quality to represent HA2, HA3 and HA4. Therefore, the algorithms do not divert from one another greatly. However, we still can distinguish them in each scenario.

Performance of the heuristics when the service time is short (10 minutes)

When service time is 10 minutes, results are illustrated in Table 5.19.

Table 5.19 Heuristic performances when service time = 10 minutes

Customer	Profits				Computational Time (ms)			
Arrival Rate	HA1	HA2	HA3	HA4	HA1	HA2	HA3	HA4
20	9.73	9.87	9.89	9.90	10	32	63	10
30	14.47	14.71	14.71	14.74	22	83	168	23
40	18.83	18.90	18.87	19.01	44	242	467	57
50	21.62	21.53	21.50	21.63	70	445	1435	189
60	23.60	23.49	23.49	23.61	102	729	2522	269
70	25.07	25.05	25.05	25.10	132	1108	3797	378
80	26.12	26.14	26.12	26.14	169	1513	5319	454

With respect to solution quality, HA4 performs the best. However, the differences among the algorithms are small, with the largest deviation from the maximum being 1.7%. The solution quality of HA2 is very close to that of HA3. When the customer arrival rate is small ($\lambda < 40$), HA1 performs not as good as the other heuristics; however, with the increase of problem size, it behaves almost as well as HA4. For computational time, HA3 takes the most time, followed by H2, H4 and H1. Meanwhile, time taken by HA4 is much more comparable to HA1 than to HA2; computational time of both HA2 and HA3 also increases much faster than the other two heuristics as arrival rate increases.

Performance of the heuristics when the service time is medium (30 minutes)

When the service time is 30 minutes, Table 5.20 summarizes both the solution quality and the computational time with the increase of customer arrival rate λ.

Table 5.20 Heuristic performances when service time = 30 minutes

Customer Arrival Rate	Profits				Computational Time (ms)			
	HA1	HA2	HA3	HA4	HA1	HA2	HA3	HA4
20	8.28	8.27	8.27	8.30	11	46	115	48
30	9.66	9.69	9.69	9.71	23	96	299	77
40	10.40	10.46	10.45	10.46	37	175	590	117
50	10.83	10.97	10.96	10.92	52	250	999	207
60	11.13	11.35	11.33	11.23	73	364	1532	274
70	11.36	11.65	11.65	11.47	95	494	2153	370
80	11.51	11.88	11.87	11.63	122	605	2794	442

With respect to solution quality, result from HA1 is always the worst in this case. For the other three heuristics, when customer arrival rate $\lambda < 50$, results are very similar to one another. However, as λ increases, both HA2 and HA3 outperform HA4 with HA2 being slightly better than HA3. The difference between the best (HA2) and the worst (HA1) also increases as λ increases with the largest difference being about 3.2%. In terms of the computational time, from largest to smallest, it follows the same order as the case when service time is short: HA3, HA2, HA4 and HA1. However, in this case, time taken by HA4 is much more comparable to HA2 than to HA1. Computational time of HA3 also increases fastest; and computational time of both HA2 and HA3 increase faster than that of HA1.

Performance of the heuristics when the service time is long (100 minutes)

When the service time is 100 minutes, results are shown in Table 5.21. Performance of HA1 and HA4 are consistently better than that of HA2 and HA3. When the customer arrival rate λ is less than 50, HA4 performs the best followed by HA1, HA3 and HA2. As λ further increases, performance of HA1 improves and it produces results very close to that of HA4; similarly, performance of HA2 also improves and it produces results slightly better than HA3. With respect to computational time, HA3 still takes the most time and HA1 takes the least time. However, in this case, HA4 becomes the second most time consuming heuristic; and HA2 becomes the second least time consuming one.

Table 5.21 Heuristic performances when service time = 100 minutes

Customer Arrival Rate	Profits				Computational time (ms)			
	HA1	HA2	HA3	HA4	HA1	HA2	HA3	HA4
20	3.521	3.492	3.504	3.535	11	24	57	41
30	3.678	3.658	3.665	3.685	19	44	120	58
40	3.758	3.742	3.745	3.761	33	68	216	87
50	3.806	3.793	3.793	3.805	53	98	345	154
60	3.838	3.828	3.824	3.840	66	142	476	193
70	3.862	3.851	3.851	3.863	89	177	639	266
80	3.877	3.870	3.867	3.877	113	206	817	310

5.7 Summary and Conclusions

In this chapter, we consider a multi-period time constrained TSP in the rolling schedule environment. The problem is to design a schedule for each period so that the average profit of serving a subset of the customers is maximized in a long run. We propose three heuristic algorithms based on iterative customer assignment (HA1, HA2 and HA3) and one heuristic based on iterative center-of-gravity scheme (HA4) for solving the problem. Then, the problem is formulated as a set-covering problem, and its LP relaxation is solved to optimality by a column generation scheme to get an upper bound for the original set-covering problem. To evaluate the performance of the proposed heuristics, for small size problem with long service times, the heuristics are compared against the upper bound; for other cases, the performance of the heuristics are compared among themselves. Computational results illustrate that, each of the best representative of the heuristics perform very well for the problem, with the largest average deviation from the upper bound being 2.24%, and the smallest deviation only 1.02%. When comparing heuristics among themselves, results indicate that, with respect to solution quality, each heuristic has its own advantage in a certain scenario. When the service time is short, the heuristics perform quite similar to one another; when the service time is medium, both HA2 and HA3 tend to behave better than HA1 and HA4; however, when the service time is long, HA1 and HA4 become better than HA2 and HA3. For the computational time, HA3 always ranks the most time consuming and HA1 ranks the least time consuming, leaving HA2 and HA4 in between. Computational time of HA4 increases much faster than that of HA2 as the service time increases: when service time is short, HA2 ranks the second time consuming; while when service time is long, HA4 becomes the second time consuming one. We make the following recommendations to the decision makers: When the service time is short and long, HA1 is the best candidate, because its solution quality is quite good compared to the other heuristics, but it takes very little computational time. When the service time is medium, both HA2 and HA3 perform very good, since HA3 takes much more time, HA2 is more preferable.

Chapter 6

The TCTSP with Stochastic Travel and Service Times

6.1 Introduction

The time constrained travelling salesman problem (TCTSP), is a variant of the classical travelling salesman problem, where only a subset of the customers can be visited due to the time limit constraint. The problem is usually considered as a deterministic problem in the literature. In practical applications, however, both the travel and the service times may not be known with certainty in advance. For example, the travel time may be affected by the weather conditions (rain or snow) as well as the traffic conditions (road repair or traffic accidents); the service time is then determined by the kind of service the customer requires. In this chapter, a TCTSP with stochastic travel and service times is considered. The problem consists of optimally selecting and sequencing a subset of customers to visit in the presence of random travel and service times to maximize the expected profit while satisfying the time limit constraint.

The problem can be formulated both as a chance-constrained program and a two stage stochastic program with recourse (SPR). In the later formulation, a subset of the customers must be optimally selected and sequenced before the particular values of the random travel and service times are known, these are called first stage decisions; once the customers are selected and their visiting order is fixed, it is possible to calculate the total travel and service time associated with the tour. Thus, in the second stage, recourse actions can be taken to impose an expected penalty on the objective function.

In practice, drivers usually are paid overtime for work done after normal hours, it is therefore reasonable to set the penalty to be proportional to the total travel and service time of the route in excess of a preset constant *T*.

The L-shaped method of Van Slyke and Wet (1969) is a cutting plane or Benders' decomposition (Benders, 1962) technique for solving the two stage stochastic linear problems when the random variables have finite support. The name 'L-shaped method' is due to the special block structure of the two stage stochastic problem in its extensive form. Birge and Louveaux (1988) extended the L-shaped algorithm with single optimality cut to a multi-cut algorithm. They showed that, the effectiveness of the multi-cut algorithm is conditional: it is more effective when the number of realizations of the random variables is not significantly larger than the number of first stage constraints. In Laporte and Louveaux (1993), they presented an integer L-shaped method for the stochastic integer program with complete recourse, in which a branching procedure is incorporated into the L-shaped method to recover the integrality of the variables. New optimality cuts were derived for the case when first stage variables are binary. These cuts are more efficient when random variables have many states or have continuous distributions, or when the second stage problem cannot be formulated in terms of first stage variables. Some lower bounds on the second stage value function were also presented in the paper. Laporte *et al.* (1992) addressed the vehicle routing problem with stochastic travel times. A branch-and-cut algorithm was proposed, and optimality cuts were generated in a similar way as in Laporte and Louveaux (1993). The integer L-shaped method has also been applied in the vehicle routing problem with stochastic demands; see Hjorring and Holt (1999), Laporte *et al.* (2002). Gendreau *et al.* (1995) applied the integer L-shaped method to the vehicle routing problem with stochastic demands and customers.

The purpose of this chapter is to present an integer L-shaped algorithm for the TCTSP with stochastic travel and service times. In Section 6.2, we give a formal description of the problem considered and formulate it as both a chance-constrained program and a stochastic program with recourse. Valid constraints used in the integer L-shaped algorithm are derived in Section 6.3. The procedure involved in the integer L-shaped algorithm is described in Section 6.4. Section 6.5 presents the computational results and analysis. Finally the results obtained in this study are summarized in Section 6.6.

6.2 Problem Description and Model Formulation

The TCTSP with stochastic travel and service times considered in this chapter can be described as follows. Let $G = (V', A)$ be a complete graph, where $V' = \{0, 1, \ldots, n\}$ is a vertex set, and $A = \{(i, j) \mid i, j \in V'$ and $i < j\}$ denotes a set of arcs. Vertex 0 represents the depot, and $1, \ldots, n$ denote n customers. Associated with each customer i, there is a profit R_i and a service time τ_i; and associated with each arc, there is a travel time t_{ij}. Assume that the travel time for each arc and the service time for each node are independent discrete random variables. Assume that T is the maximum effective working time, ΔT is the maximum amount of time allowed to exceed T. The objective is to maximize the expected profit realized from visiting a subset of the customers without violating the time limit constraint.

In stochastic programming, commonly, two versions of the problem are considered: chance-constrained program and stochastic program with recourse.

Chance-constrained program

In addition to the notations V', A, R_j, T, ΔT described above, the following notations are used in the model formulation.

$V = V' \setminus \{0\}$.

t_{ij} = A random variable representing time of travelling arc (i, j).

τ_j = A random variable representing service time of visiting node j.

α = The maximum probability that the total travel and service time within a tour is allowed to exceed $T + \Delta T$.

$$x_{ij} = \begin{cases} 1, & \text{if arc}\,(i, j) \in A\,\text{is traversed} \\ 0, & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{if node } j \in V'\,\text{is visited} \\ 0, & \text{otherwise} \end{cases}$$

The chance – constrained model is formulated as follows:

$$Z = \text{Max} \sum_{j \in V} R_j y_j \tag{6.1}$$

Subject to

$$\sum_{j \in V} x_{0j} = 2 \tag{6.2}$$

$$\sum_{(i,j) \in A} x_{ij} + \sum_{(j,l) \in A} x_{jl} = 2y_j \qquad \forall j \in V \tag{6.3}$$

$$\sum_{\substack{i \in \Lambda \\ j \in \Lambda}} x_{ij} \le |\Lambda| - 1 \qquad \forall \Lambda \subseteq V,\ |\Lambda| \ge 3 \tag{6.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A, \tag{6.5}$$

$$y_j \in \{0, 1\} \qquad \forall j \in V \tag{6.6}$$

Now consider an orienteering tour $\pi = (i_0 = 0, i_1, \ldots, i_u, i_{u+1} = i_0 = 0)$. Such a route is legal only if

$$P\left(\sum_{k=0}^{u}[t_{i_k i_{k+1}} + \tau_{i_k}] > T\right) \le \alpha \tag{6.7}$$

In the above formulation, the objective (6.1) is to maximize the total profit realized from visiting the subset of the customers. Constraints (6.2) ensure that the route must start from depot and go back to depot in the end. Constraints (6.3) indicate that, if node *j* is not included in the tour, no arcs incident to *j* are included. Otherwise, there must be one arc going into and one arc coming out of node *j*. Constraints (6.4) are sub-tour elimination constraints, which guarantee that except a sub-tour including the depot, no other sub-tours in set *V* are allowed. Constraint (6.7) removes those tours with probability that total travel and service times exceed *T* being greater than α .

Stochastic program with recourse

To present the model of stochastic program with recourse (SPR), we need slightly different notations from the chance-constrained model.

Notation:

ξ = A vector of random variables corresponding to travel and service time. It has a finite number of realizations, $\xi^1, \xi^2, ..., \xi^K$ with probabilities $p_1, p_2, ..., p_K$.

p_k = The probability that the random vector ξ takes on the realization ξ^k.

K = Number of realizations of vector ξ.

$\theta(\xi^k)$ = The total travel and service time of the route in excess of *T* when the realization of the random variable is ξ^k.

$t_{ij}^{\xi^k}$ = A random variable representing time of travelling arc (*i*, *j*) when the realization of the random variable is ξ^k.

$\tau_j^{\xi^k}$ = A random variable representing service time of visiting node *j* when the realization of the random variable is ξ^k.

β = The unit penalty cost for total time of the route in excess of *T*.

$x_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \in A \text{ is traversed} \\ 0 & \text{otherwise} \end{cases}$

$y_j = \begin{cases} 1 & \text{if node } j \in V \text{ is visited} \\ 0 & \text{otherwise} \end{cases}$

The SPR model is as follows:

$$Z = \max\left(\sum_{j \in V} r_j y_j - \beta \sum_{k=1}^{K} p_k \theta(\xi^k)\right) \tag{6.8}$$

subject to

Constraints 6.2, 6.3, 6.4, 6.5, 6.6 and

$$\theta(\xi^k) \geq \sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j \in V} \tau_j^{\xi^k} y_j - T \qquad k = 1,2,...K \tag{6.9}$$

$$0 \leq \theta(\xi^k) \leq \Delta T \qquad k = 1,2,...,K \tag{6.10}$$

In the above formulation, the objective (6.8) is to maximize the expected profit generated (include the expected penalties incurred) from visiting a subset of the *n* customers. Constraints (6.9) and (6.10) ensure that if the total travel and service time of a route is greater than *T*, the excess amount should be within ΔT.

6.3 Valid Constraints Considered in the Integer L-shaped Algorithm

In the SPR model, if we consider constraints (6.2), (6.3) and (6.4) as one block, and write constraints (6.9) and (6.10) in extensive form for each $k = 1,2,...K$, it is clear that the problem considered here has the block-angular structure, and therefore can be solved by the L-shaped method based on Benders' decomposition (Benders, 1962) technique. The basic idea of L-shaped algorithm is to approximate the term $\beta\sum_{k=1}^{K} p_k \theta(\xi^k)$ (second stage value function) in the objective function (6.8), which involves a solution of the second stage recourse linear program. This is done by building a first stage problem using an approximate term, and only evaluating the second stage value function exactly in the second stage subproblem.

The first stage problem - current problem

At a given phase of the algorithm, we call the following first stage problem as the current problem (CP).

(CP)

$$Z = \max\left(\sum_{j\in V} r_j y_j - \eta\right) \tag{6.11}$$

subject to

$$\sum_{j\in V} x_{0j} = 2$$

$$\sum_{(i,j)\in A} x_{ij} + \sum_{(j,k)\in A} x_{jk} = 2y_j \qquad \forall j \in V$$

Set of illegal route elimination constraints (6.12)

Set of optimality constraints (6.13)

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A,$$

$$y_j \in \{0,1\} \qquad \forall j \in V$$

The above current problem is obtained by two relaxations. The sub-tour elimination constraints (6.4) and second stage feasibility constraints (6.9) and (6.10) are relaxed in a number of constraints in (6.12) known as feasibility cuts. Constraints (6.9) and (6.10) are relaxed because they are not known in advance. Once we get the first stage solution, we know which of these constraints are violated. Constraints (6.4) are relaxed because, though they are known, the constraints are so numerous that it would be unrealistic to impose all of them from the beginning. Finally, the second stage expected value function $\beta\sum_{k=1}^{K} p_k\theta(\xi^k)$, is relaxed by an estimated bound η and the constraints (6.13) known as optimality cuts. Note that, in the initial first stage problem (first current problem), the constraint set (6.12) may be empty and constraint set (6.13) may only contain constraint $\eta \geq 0$. In the subsequent iteration, constraint set (6.12) includes the newly identified sub-tour elimination constraints and the second stage feasibility constraints; while constraint set (6.13) includes the newly identified optimality cuts.

Given a first stage solution (x, y, η) to the above current problem, we can get the following second stage problem, and derive feasibility and optimality cuts based on it.

The second – stage problem (SSP)

$$\min\ w = \beta\sum_{k=1}^{K} p_k\theta(\xi^k) \tag{6.14}$$

$$\theta(\xi^k) \geq \sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T \qquad k = 1,2,...K \tag{6.15}$$

$$\theta(\xi^k) \leq \Delta T \qquad k = 1,2,...,K \tag{6.16}$$

$$\theta(\xi^k) \geq 0 \qquad k = 1,2,...,K \tag{6.17}$$

Since the second stage problem is a LP problem with continuous variables, we can derive the feasibility cut and the optimality cut from the dual problem.

Similar to the application of the Benders' decomposition (Benders, 1962) method for the mixed integer program, we add both the feasibility cuts and the optimality cuts when we get an integer first stage solution, which corresponds to a set of selected customers to be visited.

The feasibility cuts

For each k, we denote I_+^k and I_-^k as dual variables corresponding to constraints (6.15) and (6.16) respectively. Then the dual problem corresponding to the above second stage problem can be described as follows:

(DSSP)

$$\text{Max } \phi = \sum_{k=1}^{K} \left(\left(\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j \in V} \tau_j^{\xi^k} y_j - T \right) \mathrm{I}_+^k - \Delta T * \mathrm{I}_-^k \right) \tag{6.18}$$

subject to:

$$\mathrm{I}_+^k - \mathrm{I}_-^k \le \beta * p_k \qquad k = 1,2,...K \tag{6.19}$$

$$\mathrm{I}_+^k \ge 0, \quad \mathrm{I}_-^k \ge 0 \qquad k = 1,2,...K \tag{6.20}$$

The above problem can be separated into K problems. For each k, $k = 1,2,...K$, we have,

(DSSP^k)

$$\text{Max } \phi^k = \left(\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j \in V} \tau_j^{\xi^k} y_j - T \right) \mathrm{I}_+^k - \Delta T * \mathrm{I}_-^k \tag{6.21}$$

subject to:

$$\mathrm{I}_+^k - \mathrm{I}_-^k \le \beta * p_k \tag{6.22}$$

$$\mathrm{I}_+^k \ge 0, \quad \mathrm{I}_-^k \ge 0 \tag{6.23}$$

If the primal second stage problem (SSP) is infeasible, then at least one of the above problems (DSSP^k) is unbounded. An extreme ray of the feasible region of the dual problem (DSSP^k) is $\boldsymbol{v_{ray}} = (\mathrm{I}_+{}^k, \mathrm{I}_-{}^k) = (1, 1)$. Since the above problem (DSSP^k) is a maximization problem, if at the direction of the extreme ray,

$\begin{pmatrix} \sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T \\ -\Delta T \end{pmatrix}' \begin{pmatrix} 1 \\ 1 \end{pmatrix} > 0$ holds, then the dual problem (DSSP^k) is unbounded and the primal second stage problem (SSP) is infeasible. To eliminate this first stage solution, we can add a feasibility constraint as follows:

$$\begin{pmatrix} \sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T \\ -\Delta T \end{pmatrix}' \begin{pmatrix} 1 \\ 1 \end{pmatrix} \le 0 \qquad (6.24)$$

To get the strongest feasibility constraint for the current first stage solution, we choose the constraint corresponding to ξ^k, which makes the total travel and service times in excess of the time limit, $\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T$, the longest, and add the constraint into constraint set (6.12) of the first stage problem (CP).

The optimality cuts

In case the first stage solution (x, y, η) is feasible, by observing the feasible region of the dual problem (DSSP^k), we know that, there are two extreme points: (0, 0) and (βp_k, 0). From the objective function of (DSSP^k), it is clear that, if the following expression holds

$$\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T \ge 0 \qquad (6.25)$$

the dual optimal solution occurs at the extreme point (βp_k, 0), an optimality cut can be derived as

$$\phi^k \ge (\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T)\beta p_k \qquad (6.26)$$

Otherwise, the optimal solution occurs at the extreme point (0, 0), and the optimal cut is

$$\phi^k \geq 0 \tag{6.27}$$

Therefore, in a single cut algorithm, we can aggregate the cuts into a single optimality cut as

$$\eta \geq \sum_{k=1}^{K} \left(\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T \right) \beta p_k \tag{6.28}$$

for all ξ^k with $\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T \geq 0, \quad k = 1,2,...K$

In a multi-cut algorithm, we add the following cuts

$$\eta \geq \sum_{k=1}^{K} \phi^k \tag{6.29}$$

$$\begin{aligned} &\phi^k \geq \left(\sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T\right)\beta p_k \quad k = 1,2,...,K \\ &if \quad \sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T \geq 0 \end{aligned} \tag{6.30}$$

$$\phi^k \geq 0 \qquad\qquad if \quad \sum_{i<j} t_{ij}^{\xi^k} x_{ij} + \sum_{j\in V} \tau_j^{\xi^k} y_j - T < 0 \tag{6.31}$$

Sub-tour elimination constraints

For the TCTSP, since we only select a subset of the customers in the solution, sub-tour elimination constraints only eliminate those sub-tours that do not contain the depot. At an integer solution, it is easy to detect a sub-tour. Once a main tour containing the depot is formed, other loops are sub-tours. Suppose we have a subset Λ = ($i_0, i_1, i_2, ..., i_u, i_{u+1} = i_0$) which forms a sub-tour, we can add the following constraint to eliminate such a sub-tour:

$$\sum_{l=0}^{u} x_{i_l i_{l+1}} \leq u \tag{6.32}$$

Since any node can be dropped from the solution in the TCTSP, we can also express the sub-tour elimination constraint in terms of the edge variables x_{ij} and the nodes variables y_i.

$$\sum_{l=0}^{u} x_{i_l i_{l+1}} \leq \sum_{i \in \Lambda \setminus \{j\}} y_i \qquad \text{for } j \in \Lambda \tag{6.33}$$

The time limit constraint added to the initial first stage problem

In the initial first stage (current) problem, after constraints (6.9) and (6.10) are relaxed, there is no time limit constraint. The feasibility cuts, playing a similar role as the time limit constraint, will only be gradually introduced into the current problem by solving the second stage problems. To avoid selecting too many customers into the first stage solution, we add the following time limit constraint to constraint set (6.12) when solving the initial first stage problem.

$$\sum_{i<j} t_{ij}^{\xi^*} x_{ij} + \sum_{j \in V} \tau_j^{\xi^*} y_j - T \leq \Delta T \tag{6.34}$$

We use the minimum realization of the random travel and service time variables to construct the vector ξ^*, because, otherwise, we may eliminate some possible better tours when forming the initial tour at the very beginning.

6.4 The Integer L-shaped Solution Method

The steps involved in the integer L-shaped algorithm can be described as follows:

Step 0: Set iteration index i : = 0. Initialize the first stage problem (the current problem) with constraint set (6.12) only containing constraint (6.34) and constraint set (6.13) only containing $\eta \geq 0$.

Step 1: Set $i = i + 1$. Solve the current problem. If the current problem has no feasible solution, go to Step 6. Otherwise, let (x^i, y^i, η^i) be the optimal solution.

Step 2: Check for existing sub-tours, if any violation is detected, add the first sub-tour elimination constraint found (Constraint 6.32 or 6.33) to the constraint set (6.12) of the current problem and go to Step 1.

Step 3: Check second stage feasibility constraints. If violated, add the most violated one (Constraint 6.24) to the constraint set (6.12) of the current problem and go to Step 1.

Step 4: Compute the value of the expected penalty for excess amount of time at the current first stage feasible solution, denote it as $\Phi(x^i, y^i)$.

Step 5: If $\eta^i \geq \Phi(x^i, y^i)$, the current problem satisfies the optimality criterion, go to step 6. Otherwise, introduce the optimality cuts (Constraint 6.28 for single cut algorithm and Constraints 6.29, 6.30 and 6.31 for multi-cut algorithm) into the constraint set (6.13) of the current problem, and go to Step 1.

Step 6: Output the best - known solution and stop.

6.5 Computational Results

To get the computational results, the algorithm is coded in *VISUAL C++*, and run on a *500 MHZ* Pentium *II* Processor with *128 MB* RAM under Microsoft Windows *98*. The MIPs are solved by calling the functions in the CPLEX optimization package into the *VC++* program.

6.5.1 Problem Data Generation

Our computational results are the averages of the results obtained from 10 randomly generated problem instances. For each problem instance, the customers are randomly generated from the square 20*20 (km^2), according to a continuous uniform distribution, with the depot situated at the center. We assume that the effective working time *T* is 480 minutes a day; ΔT , the maximum amount of time allowed to exceed *T*, is assumed to be 120 minutes. Revenue of serving a customer *j*, R_j is randomly generated from [0, 100]. The travel time between customers *i* and *j* is calculated based on the Euclidean distance d_{ij} and the speed of the vehicle. We assume that vehicle speed υ can take on 5 possibilities: 60, 50, 40, 30, 20 (km/hour). Then the travel time between any two customers *i* and *j* can be calculated according to d_{ij}/υ . The service time at each customer also takes on 5 possibilities: 10, 20, 30, 40, and 50 (minutes). We define ξ as a vector of random variables corresponding to travel and service times. It has a finite number of realizations, $\xi^1, \xi^2, ..., \xi^K$ with probabilities $p_1, p_2, ..., p_K$, with ξ^k constructed by combining the 5 possibilities of travel time and 5 possibilities of service time.

In our computational experiment, we assume $K = 5$, and constructed the data vectors for the travel time part as follows.

ξ^1: Best case; with 80% of the arcs among the customers traveling at speed 60 (km/hour), and the rest traveling at the other 4 speed possibilities.

ξ^2: Above average; with 80% of the arcs among the customers traveling at speed 50 (km/hour), and the rest traveling at the other 4 speed possibilities.

ξ^3: Average case; with 80% of the arcs among the customers traveling at speed 40 (km/hour), and the rest traveling at the other 4 speed possibilities.

ξ^4: Below average; with 80% of the arcs among the customers traveling at speed 30 (km/hour), and the rest traveling at the other 4 speed possibilities.

ξ^5: Worst case; with 80% of the arcs among the customers traveling at speed 20 (km/hour), and the rest traveling at the other 4 speed possibilities.

The service time part of the data vectors is constructed by uniformly selecting service time from the 5 possibilities: 10, 20, 30, 40 and 50 minutes.

Furthermore, we assume that the probabilities, p_1, p_2, p_3, p_4, p_5, correspond to $\xi^1, \xi^2, \xi^3, \xi^4, \xi^5$, are 0.1, 0.2, 0.4, 0.2, 0.1, respectively.

6.5.2 Computational Results and Analysis

We examine the integer L-shaped algorithm from the following aspects:

- Single optimality cut versus multiple optimality cut;
- With different unit penalty cost β;
- With different ΔT - tolerance time allowed to exceed time limit *T*;
- With different number of states of the random vector ξ.

Results are presented in Tables 6.1 to 6.4 based on the following notations.

n:	Problem size (total number of customers)
β:	Unit penalty cost for total time of the route in excess of *T*
Profit:	Profit generated from serving the customers in the solution
F-cuts:	Number of feasibility cuts generated
O-cuts:	Number of optimality cuts generated
Sub-tour:	Number of sub-tour elimination constraints generated
S:	Single optimality cut algorithm
M:	Multiple optimality cut algorithm

First, from Table 6.1, we can see that, as problem size increases, the problem becomes more difficult to solve, with more profits realized. This is due to the fact that, though we still have to satisfy the time limit constraint (the number of customers included in the solution may not change a lot), now we have more alternatives to choose from, e.g., the solution space becomes larger. Therefore, the problem difficulty increases, as indicated by the increasing number of feasibility and optimality cuts, the number of sub-tour elimination constraints, and the computational time.

Table 6.1 Average performance of the algorithm with different unit penalty cost β

N	β	Profit	F-cuts	O-cuts	Sub-tour	Time (s)
10		518.6	0.4	2.2	2.1	0.08
15		732.2	1.8	3.6	7.8	2.20
20	0.5	948.2	2.3	4.4	11.6	11.86
25		995.8	1.9	4.8	10.6	21.66
30		1081.9	2.5	4.8	26.8	372.55
35		1104.6	2.6	4.6	49.9	1083.90
10		517.2	0.4	2.2	2.0	0.25
15		704.2	1.8	3.6	8.0	0.96
20	2	890.9	2.3	4.5	19.0	25.50
25		936.5	1.9	4.7	10.5	20.84
30		1008.6	2.5	4.8	23.5	201.79
35		1021.4	2.6	4.6	64.3	1182.38

In Laporte *et al.* (1992), their computational result showed that, as the unit penalty cost increases, the difficulty of the problem also increases. This is because in their problem, all customers must be visited, and higher penalty means more penalty cuts needed and therefore more computational time taken to solve the problem. However, in the TCTSP with stochastic travel and service times, as unit penalty cost β increases, to maximize the profit realized, the number of customers visited in the solution may become lesser. There is no clear indication that difficulty of the problem increases as the unit penalty cost β increases. The computational time taken mainly depends on the number of constraints needed, especially the number of sub-tour elimination constraints.

Table 6.2 illustrates the effect of the single-cut and the multi-cut algorithm. We set unit penalty cost $\beta = 2$, and the number of states $K = 5$ in this case. Table 6.2 indicates that, the multi-cut algorithm needs to generate more optimality cuts. However, the number of sub-tour elimination constraints needed for the multi-cut algorithm is less than that needed for the single-cut algorithm. Therefore, multi-cut algorithm takes less computational time than the single cut algorithm. The multi-cut algorithm is superior to the single cut algorithm. This conforms to the findings in Birge and Louveaux (1988), though in their study, the superiority of multi-cut algorithm over the single-cut algorithm is based on the stochastic two stage linear problems.

Table 6.2 Single cut algorithm versus multi-cut algorithm

n	Single/Multiple	F-cuts	O-cuts	Sub-tour	Time (s)
10	S	0.4	1.7	2.5	0.33
	M	0.4	2.2	2.0	0.25
15	S	1.8	2.3	9.1	1.41
	M	1.8	3.6	8.0	0.96
20	S	2.3	3.4	24.2	39.62
	M	2.3	4.5	19.0	25.50
25	S	1.9	2.6	18.3	30.65
	M	1.9	4.7	10.5	20.84
30	S	2.5	3.1	28.5	235.99
	M	2.5	4.8	23.5	201.79
35	S	2.6	2.7	88.4	1688.35
	M	2.6	4.6	64.3	1182.38

In Table 6.3, we test the algorithm against the tolerance time ΔT. Intuitively, as ΔT increases, the time limit constraint becomes less restrictive, therefore, we need less number of feasibility cuts. On the other hand, as more customers may be included in the solution, the amount of time exceeded the time limit T may also increase, which may lead to more penalty incurred and therefore more optimality cuts generated. Columns F-cuts and O-cuts in Table 6.3 clearly support these facts. When the problem size is small (less than 20), as ΔT increases, the profit generated does not change much, and the computational time incurred does not increase; however, when the problem size becomes larger, with the increase of ΔT, profits generated increases, and more customers are likely to be included in the solution. Therefore, the number of sub-tour elimination constraints and the computational time also tend to increase.

Table 6.3 Average performance of the algorithm with different ΔT

N	ΔT (min)	Profits	F-cuts	O-cuts	Sub-tour	Time (s)
10	60	517.25	0.9	1.9	2.5	0.23
	120	517.25	0.4	2.2	2.0	0.25
	180	517.25	0.3	2.6	1.3	0.08
15	60	701.28	1.7	2.4	6.7	1.77
	120	704.23	1.8	3.6	8.0	0.96
	180	704.40	1.7	4.6	9.3	0.94
20	60	880.26	2.3	3.5	12.8	15.03
	120	890.92	2.3	4.5	19.0	25.50
	180	892.39	2.2	4.8	18.5	16.05
25	60	910.37	2.1	3.9	9.6	19.67
	120	936.48	1.9	4.7	10.5	20.84
	180	941.62	1.6	4.7	15.8	23.05
30	60	989.30	2.7	4.1	22.4	453.14
	120	1008.56	2.5	4.8	23.5	471.79
	180	1015.63	2.5	5.0	27.7	679.55
35	60	995.02	2.7	4.2	48.9	672.17
	120	1021.41	2.6	4.6	64.3	1182.38
	180	1033.49	2.7	4.8	73.1	5717.94

The effect of the number of states of the random vector ξ on the algorithm is shown in Table 6.4. As the number of states of ξ increases, both the number of feasibility cuts and the number of optimality cuts tend to increase. The computational time taken mainly depends on the number of sub-tour elimination constraints added. Except for problem size 20 and 25, the computational time and the difficulty of the problem increase as the number of states increases.

Table 6.4 Average performance of the algorithm with different number of states of ξ

n	States	F-cuts	O-cuts	Sub-tour	Time (s)
10	3	0.5	1.3	2.4	0.10
	5	0.4	2.2	2.0	0.25
	10	0.5	4.3	2.0	0.26
15	3	1.1	2.2	7.7	0.90
	5	1.8	3.6	8.0	0.96
	10	2.3	7.8	10.5	2.45
20	3	1.6	2.9	10.2	13.99
	5	2.3	4.5	19.0	25.50
	10	3.4	8.7	15.2	20.47
25	3	2.1	3.0	17.3	37.61
	5	1.9	4.7	10.5	20.84
	10	3.9	9.1	29.9	201.67
30	3	1.7	2.9	20.5	54.67
	5	2.5	4.8	23.5	201.79
	10	4.0	9.5	34.2	312.66
35	3	2.4	2.9	41.3	838.22
	5	2.6	4.6	64.3	1182.38
	10	4.0	8.9	94.9	23170.40

6.6 Conclusions

In this chapter, we considered the time constrained traveling salesman problem with stochastic travel and service times, which can be encountered in a number of practical situations. We formulate it as both a chance-constrained program and a two stage stochastic program with recourse, and presented an integer L-shaped algorithm for solving it. We examined the algorithm from a number of aspects. Computational results show that, for this particular problem, the difficulty mainly lies in the elimination of the sub-tours; therefore, the larger the number of sub-tours needed, the more computational time taken. Multi-cut algorithm showed its superiority to the single-cut algorithm in terms of the number of sub-tours imposed and the computational time required. As the unit penalty cost increases, difficulty of the problem does not have a clear trend of increase due to the fact that less number of customers will be visited. With the increase of the number of states of random vector ξ or the increase of the tolerance time ΔT , when the problem size is large, the difficulty of the problem increases.

Chapter 7

Conclusions and Directions of Further Research

In this chapter, first some concluding remarks are presented in Section 7.1; followed by main contributions of this thesis stated in Section 7.2; finally possible further research directions are highlighted in Section 7.3.

7.1 Summary and Conclusions

This thesis mainly addressed two variants of the classical traveling salesman problem: the vehicle routing problem with stochastic demands (VRPSD) and the time constrained traveling salesman problem (TCTSP).

In thesis Part I, we studied the VRPSD based on the formulation of stochastic programming with recourse, which is within the framework of *a priori* optimization. A comparative study among heuristics available in the literature was firstly carried out to determine which one is superior to the others in a certain context; and valuable suggestions and recommendations were made for decision makers in various scenarios. As most of the heuristics presented in the literature belong to classical local search algorithms, the thesis also examined whether metaheuristics are more preferable for the VRPSD. Three metaheuristics, such as simulated annealing (SA), threshold accepting (TA) and tabu search (TS), were presented for solving the VRPSD. Comprehensive computational experiment was carried out to compare the performance of the proposed metaheuristics with that of the heuristics presented in the literature in various situations with respect to problem size and demand pattern. Results show that, metaheuristics can provide solutions with better solution quality for VRPSD, though they may consume more computational time. In some cases, the time taken by metaheuristics even can be comparable to classical local search methods. For example, for multiple vehicle case, when the problem size is relatively large, SA can provide better solutions with similar or less computational time than the R1-C2 in Yang *et al.* (2000). Chapter 4 also carried out a performance comparison among the metaheuristics proposed for the VRPSD to determine which one is superior to the others in a certain context, and provide the decision makers with more choices and more valuable suggestions.

In the literature, most researchers address the single tour TCTSP or orienteering problem (OP). Part II of this thesis firstly studied a multi-period TCTSP: The problem is to find a subset of customers as well as the sequence of serving them in each period in a rolling schedule environment, so that the average profit per period in the long run is maximized. Several heuristics based on iterative customer assignment and iterative centre-of-gravity were proposed for solving the problem. Then, a set-covering type formulation was presented for the problem within an elongated rolling horizon, and its LP relaxation was solved to optimality by a column generation scheme to get an upper bound for the original problem. To evaluate the performance of the heuristics, for small size problem with long service times, the heuristics were compared against the upper bound; for other cases, the performances of the heuristics were compared among themselves. Computational results illustrate that, the best representative of each heuristic performs very well for the problem, with the largest average percentage deviation from the upper bound being 2.24%, and the smallest deviation only 1.02%. When comparing the heuristics among themselves, results indicate that, with respect to solution quality, each heuristic has its own advantage in a certain scenario. Decision makers are advised to employ different heuristics in different scenarios.

For TCTSP, due to the effective working time limit constraint, the profit realised from visiting the subset of the customers is directly affected by the travel time and service time, which are most unlikely to be known with certainty in advance due to their dynamic nature. However, most literature considers the deterministic TCTSP. The second problem considered in Part II of this thesis is a stochastic TCTSP: an extension of the TCTSP into a stochastic case, where the travel and service times are assumed to be random variables. Again, within the framework of *a priori* optimization, two models: a chance-constrained program and a stochastic program with recourse, were proposed for formulating the problem. Then an integer L-shaped solution method was developed to solve the problem to optimality. The computational experiment examined the algorithm in several scenarios. Results show that, the algorithm based on integer L-shaped method can solve the stochastic TCTSP with moderate problem size to optimality within reasonable amount of time.

7.2 Main contributions of this study

For Part I of this thesis, the comparative study helps us to identify heuristics with better performance in a certain context, so that we can provide valuable suggestions to the practitioners. Secondly, our study on the performance of the three metaheuristics, the simulated annealing (SA), threshold accepting (TA), and tabu search (TS) algorithms for the VRPSD, not only answers the question whether metaheuristics are suitable for solving the VRPSD, but also provides more choices and more valuable suggestions to the practitioners.

For Part II of this thesis, the study on the multi-period TCTSP has both theoretical and practical significance. We provide a systematic study of this problem, as it is seldom studied in the literature. This includes:

- We incorporate the concept of rolling schedule into the study of the problem, which can nicely capture the dynamic nature of the customer information: when a customer needs a service.
- We present a set-covering type formulation for the problem within one rolling horizon, and a column generation solution method to find an upper bound for this problem. This type of formulation and the column generation solution method can be applied to similar problems, such as the team orienteering problem (Chao *et al*. 1996b), to find the optimal solution or an upper bound of the problems, so that they can be employed as performance measurers to evaluate the heuristics provided in the literature.
- We provide several efficient heuristic methods with good performance in terms of both solution quality and computational time for this kind of problem. Moreover, the heuristics are studied in detail under different problem parameter settings, so that suggestions and recommendations in different scenarios can be made for potential applications and therefore a guideline can be provided for the decision makers in their decision process.

For the second problem in Part II, our focus is to extend the deterministic TCTSP into the stochastic case with travel and service times being considered as random variables, because they are the problem parameters that are most unlikely to be known for sure in advance; and also they are very important in the TCTSP, as they directly affect the solution and therefore the profits generated from the solution. However, for the TCTSP, or even in a much broader context, the vehicle routing problem, studies focused on the stochastic travel and service times are quite few. Though Laporte *et al*. (1992) addressed the vehicle routing problem with stochastic travel times, it is a two stage stochastic problem with complete recourse: the first stage solution is always second stage feasible, though penalty may occur in the second stage problem. For our problem, on the other hand, due to the constraint that the total travel and service times of a tour in excess of time limit T cannot exceed ΔT, second stage feasibility is also a main concern. Therefore, in addition to optimality cuts, feasibility cuts must also be considered in the second stage problem, which adds more difficulty to the problem considered.

7.3 Directions of Further Research

(1) Further research directions for VRPSD

For the VRPSD with multiple vehicles, in this study as well as in Yang *et al.* (2000), to partition the customers into several routes, the constraint imposed is that the expected route length should be within a predefined limit. It would be more accurate and reasonable if the variance of the route length could be taken into consideration as well.

For the tabu search metaheuristic, the current study only examined its performance on the single vehicle case; it would be worthwhile to develop TS algorithm for the VRPSD with multiple vehicles. Moreover, one direction of further research regarding metaheuristics is to study how some other recently developed metaheuristics, such as ant algorithms, can be applied to the VRPSD.

(2) Further research directions for TCTSP

For the TCTSP, one possible direction is to extend the deterministic multi-period TCTSP in a rolling schedule environment to the stochastic case with stochastic service and/or travel times, and develop effective algorithms for solving the problem.

Furthermore, from the computational results presented in Chapter 6, it is clear that, even for the single period TCTSP with stochastic travel and service times, the exact algorithm is computationally expensive. Therefore, for large size stochastic TCTSP, especially stochastic multi-period TCTSP in the rolling schedule environment, it is important to design and develop heuristics with good performance for this kind of problems.

One prerequisite for developing heuristics is how to evaluate a tour and therefore select the best possible one. For a deterministic problem, it is an easy task; however, when travel and service times are stochastic variables, it is nontrivial. In the Appendix, the thesis also presented how to evaluate a single period TCTSP with stochastic service times, given that we employ the total profit (including the expected penalty incurred) realized from such a TCTSP with stochastic service times as the performance measurer. Clearly, as one visit more customers, the time in excess of the effective working time limit T increases; therefore the penalty incurred increases as well. One possible research direction is to investigate whether there are rules regarding how to select the more profitable customers and when to stop visiting the less profitable customers.

References

[1] Aarts, E. and Lenstra, J. K. (1997). Local search in combinatorial optimization. New York: Wiley.

[2] Alfa, A.S., Heragu, S.S. and Chen, M. (1991). A 3-opt based simulated annealing algorithm for vehicle routing problem. *Computers and Industrial Engineering* 21, 635-639.

[3] Arabeyre, J.P. *et al*. (1969). The airline crew scheduling problem: A survey. *Transportation Science* 3 (2), 140-163.

[4] Awerbuch, B., Azar, Y., Blum, A., and Vempala, S. (1998). New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing* 28 (1), 254-262.

[5] Baker, E.K. *et al*. (1979). Efficient heuristic solution to an airline crew scheduling problem. *AIIE Transactions*, 79-84.

[6] Balas, E. (1965). An additive algorithm for solving linear programs with zero-one variables. *Operations Research* 13, 517-549.

[7] Balas, E. (1975). Some valid inequalities for the set partitioning problem. MSRR 368, Carnegie-Mellon University, Pittsburgh, PA.

[8] Balas, E. (1989). The prize collecting traveling salesman problem. *Networks* 19 (6), 621-636.

[9] Balas, E. (1995). The prize collecting traveling salesman problem: II. Polyhedral Results. *Networks* 25 (4), 199-216.

[10] Balas, E. and Martin, C.H. (1991). Combinatorial optimization in steel rolling (extended abstract). Workshop on Combinatorial Optimization in Science and Technology (COST), RUTCOR.

[11] Balas, E. and Padberg, M. (1975). On the set-covering problem: II. An algorithm for set partitioning. *Operations Research* 23 (1), 74-90.

[12] Balas, E. and Samuelson, H. (1974). A symmetric subgradient cutting plane method for set partitioning. W.P. 5-74-75, Carnegie-Mellon University, Pittsburgh, PA.

[13] Balinski, M.L. and Quandt, R.E. (1964). On an integer program for a delivery problem. *Operations Research* 12, 300-304.

[14] Bartholdi, J.J. and Platzman, L.K. (1982). An $O(n \log n)$ planar traveling salesman heuristic based on spacefilling curves. *Operations Research Letters* 1, 121-125.

[15] Bartholdi, J.J., Platzman, L.K., Collins, R.L. and Warden W.H. (1983). A minimal technology routing system for meals on wheels. *Interfaces* 13, 1-8.

[16] Benders, J.F. (1962). Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik* 4, 238-252.

[17] Bentley, J.L. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing* 4, 387-411.

[18] Benton, W.C. and Rossetti, M.D. (1992). The vehicle scheduling problem with intermittent customer demands. *Computers & Operations Research* 19, 521-531.

[19] Bertsimas, D.J. (1988). Probabilistic combinatorial optimization problems. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[20] Bertsimas, D.J., Jaillet, P. and Odoni, A. (1990). A priori optimization. *Operations Research* 38, 1019-1033.

[21] Bertsimas, D.J. (1992). A vehicle routing problem with stochastic demand. *Operations Research* 40, 574-585.

[22] Bertsimas, D.J., Chervi, P. and Peterson, M. (1995). Computational approaches to stochastic vehicle routing problems. *Transportation Science* 29, 342-352.

[23] Bienstock, D., Goemans, M.X., Simchi-Levi, D. and Williamson, D. (1993). A note on the prize collecting traveling salesman problem. *Mathematical Programming* 59 (3), 413-420.

[24] Birge, J.R. and Louveaux, F.V. (1988). A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research* 34, 384-392.

[25] Bland, R.G. and Shallcross, D.F. (1989). Large traveling salesman problems arising from experiments in X-ray crystallography: a preliminary report on computation. *Operations Research Letters* 8, 125-128.

[26] Bodin, L., Golden, B.L., Assad, A. and Ball, M. (1983). Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research* 10, 63-211.

[27] Busacker, R.G. and Saaty, T.L. (1965). Finite graphs and networks. McGraw-Hill, New York.

[28] Carraway, R.L., Morin, T.L. and Moskowitz, H. (1989). Generalized dynamic programming for stochastic combinatorial optimization. *Operations Research* 37, 819-829.

[29] Chao, I.M., Golden, B.L. and Wasil, E.A. (1996a). A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research* 88 (3), 475-489.

[30] Chao, I.M., Golden, B.L. and Wasil, E.A. (1996b). The team orienteering problem. *European Journal of Operational Research* 88 (3), 464-474.

[31] Christofides, N. (1974). The vehicle routing problem. In: NATO Conference on Combinatorial Optimization, Paris.

[32] Christofides, N. (1976). Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388. Graduate School of Industrial Administration. Carnegie Mellon University, Pittsburgh, PA.

[33] Clarke, G. and Wright, J.W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research* 12 (4), 568-581.

[34] Cloonan, J. (1966). A heuristic approach to some sales territory problems. In: Little, JDC (Eds.), Proceedings of the fourth international conference on Operations Research. Cambridge, Massachusetts: MIT Press, 81-84.

[35] Croes, G.A. (1958). A method for solving traveling salesman problems. *Operations Research* 6, 791-812.

[36] Dantzig, G.B. and Ramser, J.H. (1959). The truck dispatching problem. *Management Science* 6 (1), 80-91.

[37] Dror, M. (1993). Modeling vehicle routing with uncertain demands as a stochastic program: Properties of the corresponding solution. *European Journal of Operational Research* 64, 432-441.

[38] Dror, M., Ball, M. O. and Golden, B. L. (1985). Computational comparison of algorithms for inventory routing. *Annals of Operations Research* 4, 3-23.

[39] Dror, M., Laporte, G. and Louveaux, F.V. (1993). Vehicle routing with stochastic demands and restricted failures. *Zeitschrift für Operations Research* 37, 273-283.

[40] Dror, M., Laporte, G. and Trudeau, P. (1989). Vehicle routing with stochastic demands: properties and solution frameworks. *Transportation Science* 23 (3), 166-176.

[41] Dueck, G. and Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics* 90(1), 161-175.

[42] Duhamel, C., Potvin, J.Y. and Rousseau, J.M. (1997). A tabu search heuristic for the vehicle routing problem with backhauls and time windows. *Transportation Science* 31, 49-59.

[43] Fiechter, C.N. (1994). A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics* 51, 243-267.

[44] Fisher, M.L. (1996). Vehicle routing. In: Networks and Distribution, Handbooks in Operations Research and Management Science. North-Holland, Amsterdam.

[45] Garfinkel, R.S. and Nemhauser, G.L. (1969). The set-partitioning problem: Set covering with equality constraints. *Operations Research* 17 (5), 848-856.

[46] Gendreau, M., Hertz, A. and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science* 40 (10), 1276-1290.

[47] Gendreau, M., Laporte, G. and Seguin, R. (1995). An exact algorithm for the vehicle routing problem with stochastic customers and demands. *Transportation Science* 29, 143-155.

[48] Gendreau, M., Laporte, G. and Seguin, R. (1996a). A Tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research* 44, 469-477.

[49] Gendreau, M., Laporte, G. and Seguin, R. (1996b). Stochastic vehicle routing. *European Journal of Operational Research* 88, 3-12.

[50] Gendreau, M., Laporte, G. and Semet, F. (1998a). A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks* 32 (4), 263-273.

[51] Gendreau, M., Laporte, G. and Semet, F. (1998b). A tabu search heuristic for the undirected selective traveling salesman problem. *European Journal of Operational Research* 106 (2-3), 539-545.

[52] Gendreau, M., Guertin, F., Potvin, J.Y. and Taillard, E. (1999). Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science* 33 (4), 381-390.

[53] Gensch, D.H. (1978). An industrial application of the traveling salesman's sub-tour problem. *AIIE Transactions* 10(4), 362-370.

[54] Glover, F. and Laguna, M. (1997). Tabu Search. Kluwer Academic Publishers, Boston.

[55] Golden, B.L., Levy, L. and Dahl, R. (1981). Two generalizations of the traveling salesman problem. *Omega* 9, 439-445.

[56] Golden, B.L., Levy, L. and Vohra, R. (1987). The orienteering problem. *Naval Research Logistics* 34, 307-318.

[57] Golden, B.L. and Assad, A. (1988). Vehicle routing: methods and studies. North-Holland, Amsterdam.

[58] Golden, B.L., Wang, Q. and Liu, L. (1988). A multifaceted heuristic for the orienteering problem. *Naval Research Logistics* 35(3), 359-366.

[59] Gomory, R.E. (1963). An algorithm for integer solutions to linear programs. In: Graves, S.C., Wolfe, P.M. (Eds.). *Recent Advances in Mathematical Programming*. McGraw-Hill, New York.

[60] Haimovitch, M. and Rinnooy Kan, A. (1985). Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research* 10, 527-542.

[61] Hiquebran, D.T., Alfa, A.S., Shapiro, J.A. and Gittoes, D.H. (1994). A revised simulated annealing and cluster-first route-second algorithm applied to the vehicle routing problem. *Engineering Optimization* 22, 77-107.

[62] Hjorring, C. and Holt, J. (1999). New optimality cuts for a single–vehicle stochastic routing problem. *Annals of Operations Research* 86, 569-584.

[63] Jaillet, P. (1985). Probabilistic traveling salesman problem. Ph.D. thesis, Report No. 185, Operations research Center, Massachusetts Institute of Technology, Cambridge, MA.

[64] Jaillet, P. (1988). A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Operations Research* 36, 929-936.

[65] Kantor, M.G. and Rosenwein, M.B. (1992). The orienteering problem with time windows. *Journal of the Operational Research Society* 43 (6), 629-635.

[66] Kao, E.P.C. (1978). A preference order dynamic program for a stochastic traveling salesman problem. *Operations Research* 26, 1033-1045.

[67] Kataoka, S., Yamada, T. and Morito, S. (1998). Minimum directed 1-subtree relaxation for score orienteering problem. *European Journal of Operational Research* 104 (1), 139-153.

[68] Keller, C.P. (1985). Multiobjective routing through space and time: The MVP and TDVP problems. Ph.D. Dissertation, The University of Western Ontario, London.

[69] Keller, C.P. (1989). Algorithms to solve the orienteering problem: a comparison. *European Journal of Operational Research* 41 (2), 224-231.

[70] Kirkpatrick, S., Gerlatt, C.D.Jr. and Vecchi, M.P. (1983). Optimization by simulated annealing. *Science* 220, 671-680.

[71] Korte, B.H. (1989). Applications of combinatorial optimization. In: M. Iri, K. Tanabe (Eds.). Mathematical programming: Recent developments and applications. Kluwer, Dordrecht. p. 1-55.

[72] Korte, B.H. and Vygen, J. (2000). Combinatorial optimization: theory and algorithms. Berlin: Springer.

[73] Lambert, V., Laporte, G. and Louveaux, F. V. (1993). Designing collection routes through bank branches. *Computers & Operations Research* 20, 783-791.

[74] Laporte, G. and Martello, S. (1990). The selective traveling salesman problem. *Discrete Applied Mathematics* 26, 193-207.

[75] Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 345-358.

[76] Laporte, G., Louveaux, F.V. and Mercure, H. (1992). The vehicle routing problem with stochastic travel times. *Transportation Science* 26, 161-170.

[77] Laporte, G. and Louveaux, F.V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters* 13, 133-142.

[78] Laporte, G., Louveaux, F.V. and Mercure, H. (1994). A priori optimization of the probabilistic traveling salesman problem. *Operations Research* 42, 543-549.

[79] Laporte, G., Louveaux, F.V. and Hamme, Luc Van. (2002). An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research* 50, 415-423.

[80] Larson, R.C. (1988). Transporting sludge to the 106-mile site: An inventory routing algorithm for fleet sizing and logistic system design. *Transportation Science* 22, 186-198.

[81] Lawler, E.L., Lenstra, J.K., Rinnooy, A.H.G. Kan and Shmoys, D.B. (eds.) (1985). The traveling salesman problem: A guided tour of combinatorial optimization. John Wiley, Chichester, U.K.

[82] Leifer, A.C. and Rosenwein, M.B. (1994). Strong linear-programming relaxations for the orienteering problem. *European Journal of Operational Research* 73 (3), 517-523.

[83] Levin, A. (1969). Fleet routing and scheduling problem for air transportation system. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA.

[84] Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal* 44, 2245-2269.

[85] Lopez, L., Carter, M.W. and Gendreau, M. (1998). The hot strip mill production scheduling problem: A tabu search approach. *European Journal of Operational Research* 106 (2-3), 317-335.

[86] Martello, S. and Toth, P. (1981). An algorithm for the generalized assignment problem. In: Brans, J.P. (Eds.), Operational Research' 81, North-Holland, Amsterdam, 589-603.

[87] Mehrotra, A., Murphy, K.E. and Trick, M.A. (2000). Optimal shift scheduling**:** A branch-and-price approach. *Naval Research Logistics* 47(3), 185-200.

[88] Mittenthal, J. and Noon, C.E. (1992). An insert/delete heuristic for the traveling salesman subset-tour problem with one additional constraint. *Journal of the Operational Research Society* 43 (3), 277-283.

[89] Moreland, J.A. (1966). Scheduling of airline flight crews. Master’s Thesis, Department of Aeronautica and Astronomics, Massachusetts Institute of Technology, Cambridge, MA.

[90] Ong, H. L., Ang, B. W., Goh, T. N. and Deng, C. C. (1997). A vehicle routing and scheduling problem with time windows and stochastic demand constraints. *Asia-Pacific Journal of Operational Research* 14, 1-17.

[91] Or, I. (1976). Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking, Ph.D. thesis, Evanston, IL: Northwestern University.

[92] Osman I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research* 41, 421-451.

[93] Pierce, J.F. (1968). Application of combinatorial programming to a class of all-zero-one integer programming problems. *Management Science* 15, 191-209.

[94] Pierce, J.F. (1970). Pattern sequencing and matching in stock cutting operations. *Tappi* 53 (4), 668-678.

[95] Potvin, J.Y., Kervahut, T., Garcia, B.L. and Rousseau, J.M. (1996). The vehicle routing problem with time windows－Part I: Tabu search. *INFORMS Journal on Computing* 8 (2), 158-164.

[96] Protonotarios, M., Mourkousis, G., Vyridis, I. and Varvarigou, T. (2000). Very large scale vehicle routing with time windows and stochastic demand using genetic algorithms with parallel fitness evaluation. *Lecture Notes in Computer Science* 1823, 467-476.

[97] Ramesh, R., Yoon, Y.S. and Karwan, M.H. (1992). An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing* 4, 155-165.

[98] Revelle, C. *et al*. (1970). An analysis of private and public sector location models. *Management Science* 16 (12), 692-707.

[99] Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M. (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing* 6, 563 – 581.

[100] Salveson, M.E. (1955). The assembly line balancing problem. *Journal of Industrial Engineering* 6 (3), 18-25.

[101] Sarin, S.C. and Aggarwal, S. (2001). Modeling and algorithmic development of a staff scheduling problem. *European Journal of Operational Research* 128(3), 558-569.

[102] Savelsbergh, M.W.P. and Goetschalckx, M. (1995). A comparison of the efficiency of fixed versus variable vehicle routes. *Journal of Business Logistics* 16, 163-188.

[103] Savelsbergh, M. and Sol, M. (1998). Drive**:** Dynamic routing of independent vehicles. *Operations Research* 46, 474-490.

[104] Secomandi, N. (1998). Exact and heuristic dynamic programming algorithms for the vehicle routing problem with stochastic demands. Ph.D. thesis. University of Houston.

[105] Sniedovich, M. (1981). Analysis of a preference order traveling salesman problem. *Operations Research* 29, 1234-1237.

[106] Solomon, M.M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* 35, 254-265.

[107] Spitzer, M. (1961). Solution to the crew scheduling problem. Presented at the First AGIFORS Symposium.

[108] Stewart, W.R. Jr. and Golden, B.L. (1983). Stochastic vehicle routing: a comprehensive approach. *European Journal of Operational Research* 14, 371-385.

[109] Taillard, E. (1993). Parallel iterative search methods for vehicle routing problems. *Networks* 23, 661-673.

[110] Taillard, E., Badeau, P., Gendreau, M., Guertin, F. and Potvin, J.Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science* 31 (2), 170-186.

[111] Teng, S.Y., Ong, H.L. and Huang, H.C. (2001). A comparative study on algorithms for vehicle routing problems with stochastic demands. In: Proceedings of the 5th International Conference on Optimization: Techniques and Applications, Hong Kong, 1621-1629.

[112] Teodorovic, D. and Pavkovic, G. (1992). A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demand. *Transportation Planning and Technology* 16, 261-273.

[113] Tillman, F. (1969). The multiple terminal delivery problem with probabilistic demands. *Transportation Science* 3, 192-204.

[114] Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society* 35(9), 797-809.

[115] Van Slyke, R.M. and Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics* 17, 638-663.

[116] Waters, C.D.J. (1989). Vehicle-scheduling problems with uncertainty and omitted customers. *Journal of the Operational Research Society* 40, 1099-1108.

[117] Winker, P. (2000). Optimization Heuristics in Econometrics: Applications of Threshold Accepting. Wiley, New York.

[118] Yang, W.H., Mathur, K. and Ballou, R.H. (2000). Stochastic vehicle routing problem with restocking. *Transportation Science* 34, 99-112.

Appendix

Define the problem on a complete graph $G = (V', A)$, where V' = {0, 1,…, n} is a vertex set, and A = {(i, j) | i, $j \in V'$ and $i < j$} denotes a set of arcs. Vertex 0 represents the depot, and 1, …, n denote n customers. Associated with each customer i, there is a profit R_i and a service time τ_i; and associated with each arc, there is a travel time t_{ij}. Assume that the travel time for each arc is deterministic, while the service times for the customers are random variables. Here, we assume that all τ_i, i = 1, 2,…, n are discrete independent random variables with a known probability distribution. Assume that T is the maximum effective working time. β is the unit penalty cost for total time of the route in excess of T. The objective is to maximize the total profit realized from visiting a subset of the customers without violating the time limit constraint.

For the problem considered, we employ the total profit realized from such a TCTSP with stochastic service times as the performance measurer, which can be recursively calculated as described below.

Let t denote the available remaining time to travel. Assume that the stochastic service time τ_i of customer i follows a discrete distribution with K possible values: ξ^1, ξ^2, …, ξ^K. Let $p_i(k)$ be $P(\tau_i = \xi^k)$, the probability that service time at node i is ξ^k. Let S_j be the set of all possible states (available remaining time) in stage j. $f_j(t)$ is the profit from depot to node j when the state is t. $\Pr_j(t)$ is the probability that the state at node j is t. $\varphi_j(t)$ is the penalty incurred at node j when the state is t.

Initialization:

$$f_0(T) = 0 \tag{A.1}$$

$$\Pr_0(T) = 1 \tag{A.2}$$

Recursion:

$$f_j(t) = R_j + \sum_{k:t+t_{j-1j}+\xi^k \in S_{j-1}} (f_{j-1}(t + t_{j-1j} + \xi^k) - \varphi_j(t))\, p_j(k) \tag{A.3}$$

Then the penalty function $\varphi_j(t)$ is as follows.

$$\varphi_j(t) = \begin{cases} 0 & t \geq 0 \\ -\beta\, t & t < 0 \text{ and } t + t_{j-1j} + \xi^k \geq 0 \\ \beta(t_{j-1j} + \xi^k) & t + t_{j-1j} + \xi^k < 0 \end{cases} \tag{A.4}$$

The probability of the state *t* at stage *j*:

$$\Pr_j(t) = \sum_{k:t+t_{j-1j}+\xi^k \in S_{j-1}} \Pr_{j-1}(t + t_{j-1j} + \xi^k)\, p_j(k) \tag{A.5}$$

Last stage (go back to depot):

$$profit = \sum_{t \in S_{n-1}} (f_{n-1}(t) - \varphi_n(t)) \Pr_{n-1}(t) \tag{A.6}$$

The penalty function $\varphi_n(t)$ is as follows.

$$\varphi_n(t) = \begin{cases} 0 & t - t_{n-1n} \geq 0 \\ \beta(t - t_{n-1n}) & t > 0 \text{ and } t - t_{n-1n} < 0 \\ \beta\, t_{n-1n} & t < 0 \end{cases} \tag{A.7}$$