Heuristic Methods For Vehicle Routing Problem with Time Windows

Submited by

ZHU QILI, KENNY
Department of Electrical Engineering

As a Partial Fulfillment of the Requirements for
the Bachelor Degree of Engineering (Honors)
National University of Singapore

May 12, 1999

Abstract

This thesis documents the author's research of Vehicle Routing Problem with Time Windows (VRPTW) during his final year project. The objective of the problem is to service a number of customers within predefined time windows at minimum cost, without violating the capacity and total trip time constraints for each vehicle. Combinatorial optimization problems of this kind are NP-hard and are best solved by heuristics. Several heuristic methods, namely Simulated Annealing, Tabu Search and Genetic Algorithm, have been developed and implemented in this project. These methods were then applied to 56 Solomon's VRPTW instances and yielded competitive results as opposed to the best known solutions in the literature. This project was jointly supervised by Assistant Professor Tan Kay Chen from Department of Electrical Engineering and Assistant Professor Lee Loo Hay from Department of Industrial and Systems Engineering, and has been technically sponsored by Hewlett Packard Singapore Pte Ltd.

In the first chapter, the author presents an introduction to Vehicle Routing Problem (VRP), its position in Operating Research and Computer Science, its complexity, and the objectives of this project. The second chapter presents some of the many theoretical works appearing in the related literature. The problem definition and its mathematical model follow in Chapter 3. A brief description of mathematical programming in application of VRPTW is also included in Chapter 4. In Chapter 5, 6, 7 and 8, the details of four heuristic methods are discussed, namely λ-interchange Local Search Descent method, Simulated Annealing, Tabu Search and Genetic Algorithm. The experiment results of these methods are presented and comparisons made in Chapter 9. A description of a test 12-customer problem, complete detailed algorithms and various tables of comparison can be found in the appendices.

Acknowledgment

This thesis has benefited from the following people: My beloved mom and dad; my project supervisors, Dr. Tan Kay Chen, Dr. Lee Loo Hay, who gave me indispensable guidance and advises both academically and personally. Thank you people. Without you this thesis can never be materialized. I would also like to thank my buddies Jiang Mingliang and Khoh Chee Jeun for giving me all the courage, inspiration and the joy. I have also received precious help from various research students and engineers, especially June Chandra, Khor Eik Fun, Daniel, at Control and Simulation Laboratory. I would also like to express my gratitude to the lab technicians at Control and Simulation Lab for providing me with a fast PC to run my programs all day, all night!

The following products/software/programs/concepts were used in this project, and they belong to their respective owners:

- IBM Thinkpad is a trademark of International Business Machines
- HP Deskjet 695C is a product by Hewlett Packard Co.
- Linux Kernel 2.0.30, 2.0.36 belong to Linus Torvalds
- GNU C++ Compiler, GDB, and various GNU applications are registered software of GNU
- LaTeX is a typesetting program written and registered by Leslie Lamport
- LyX was written and registered by Matthias Ettrich, et al.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Logistics may be defined as "the provision of goods and services from a supply point to various demand points"[12]. A complete logistic system involves transporting raw materials from a number of suppliers or vendors, delivering them to the factory plant for manufacturing or processing, movements of the products to various warehouses or depots and eventually distribution to customers. Both the supply and distribution procedure require effective transportation management. Good transportation management can practically save a private company a considerable portion of its total cost. Potential cost savings constitute: lowered trucking cost due to more optimal routes and shorter distances, reduced in-house space and related costs, less penalty incurred due to untimely delivery.

Effective transport management poses a variety of problems at all three levels of strategic, tactical and operational planning[5]. Locations of the depots and the vendors may be categorized as strategic, and the problems of fleet size, mix loading, etc. may be considered tactical, while at the operational level, optimizing of routing and scheduling is concerned. This paper deals with the final level, i.e. the operational level of the distribution decision-making, especially vehicle routing decisions.

1.2 Classification of Vehicle Routing Problems

Basic vehicle routing problems can be stated as such: Given a set of nodes and/or arcs to be serviced by an unrestricted number of vehicles, construct the routes for these vehicles that are both feasible and low cost[5]. However this problem can be refined in many ways. There can be a single depot in the system or multiple depot; there can be only one vehicle or multiple vehicles; time and capacity constraints can be imposed on the nodes and the vehicles under consideration, etc. With these parameters in mind, we can construct the following table that shows the classification of Vehicle Routing Problems according to a variety of criteria.

CHARACTERISTICS	POSSIBLE OPTIONS
1. Size of available fleets	one vehicle multiple vehicles
2. Types of available fleets	homogeneous (one vehicle type) heterogeneous (multiple vehicle types)
3. Housing of vehicles	single depot multiple depot
4. Nature of demands	deterministic (known) demands stochastic demand requirements partial satisfaction of demand allowed
5. Location of Demands	at nodes on arcs mixed
6. Vehicle capacity restrictions	imposed (same) imposed (different vehicle capacities) not imposed
7. Maximum route time	imposed (same for all routes) imposed (different for different routes) not imposed
8. Time windows	single-sided double-sided

Table 1.1: Classification of VRP

Classical routing problems like Traveling Salesman Problem (TSP)and Chinese Postman Problem (CPP) can be classified as node routing problem and arc routing problem respectively. In this project, we are concerned with vehicle routing problem with time windows, one of the most encountered yet difficult node routing problems.

1.3 Computation Complexity of Vehicle Routing Problems

An important consideration in the formulation and solution of various vehicle routing problems is the computation complexity and burden associated with the techniques for these problems. Naturally, computation burden increases with the size of the problem, e.g. number of nodes in the traffic network and constraints imposed. If the computation intensity increases too rapidly, it is impractical to find a best solution even for problems of moderate sizes, given the computation power today. It is to the best interest of the users if the VRP is solved by a "*polynomially-bounded*" algorithm[5], that is a procedure whose computational burden increases only polynomially with the problem size. Decision problems which can be solved by such algorithms are said to be P problems. Unfortunately, VRP does not belong to this class of problems most of the time, and they are usually in class *NP*. Traveling Salesman Problem (TSP) has been proved to be *NP-complete*. VRPs which can be extended from TSP and has not been proved NP-complete are called *NP-hard*. NP-hard problems take exponentially or factorially increasing amount of time to solve using conventional mathematical methods, such as *integer programming (IP)*. VRPTW is such an example. It is estimated that an average VRPTW has $N^2 \times K$ decision variables, where N is the number of nodes including the depot and K is the number of vehicles used. An exhaustive search has to investigate $2^{N^2 \cdot K}$different combinations trying to satisfy a total of $N^2K + NK + 3K + N + 1$ constraints[1]. This is a huge number even if we are talking about a simple 6-node problem. Faced with NP-hard problems, one usually resorts to heuristics or approximate procedures that give an optimal or near optimal solution in tolerable time. In many cases, it is possible to obtain a bound of the deviation of the solution to the optima in the worse case. In this project, we are trying to apply various heuristic methods to VRPTWs and the worst case solution can be found using *Push-Forward Insertion Heuristic (PFIH)*[41]. With refined heuristics VRPTWs of 100 nodes can be solved in maximum of 1 hour and minimum of merely 10 seconds, according to the author's experience.

[1] See Chapter 3.

1.4 Objectives of the Project

This project was originally inspired by the vehicle routing optimization project carried out at Hewlett Packard Singapore Pte. Ltd. during the author's internship with the company. The objectives of this project are to compile a summary of the Vehicle Routing Problem (VRP) methodologies today, evaluate several existing heuristic algorithms to tackle Vehicle Routing Problems with Time Windows (VRPTW), design and implement new VRPTW optimizing techniques based on the methods studied, and finally, compare and contrast the results obtained with published results and analyze the results.

Chapter 2

Literature Review

The author has done an extensive literature review of the VRPTW during this project, mainly in the area of the concepts of VRPTWs, OR algorithms in solving VRPTWS and heuristics. In the following sections, our findings will be presented subsequently.

2.1 VRPTW Concepts

According to Thangiah, et. al.[43], Vehicle Routing Problem with Time Windows (VRPTW) involves routing a fleet of vehicles, with limited capacities and travel times, from a central depot to a set of geographically dispersed customers with known demands within specified time windows. VRPTW is obviously an extension of normal VRPs, which do not have time or load constraints. In VRPTW, one vehicle leaves its depot, travels to different nodes or pickup points within the time window each node specifies and pick up goods until the vehicle can no longer capacitate any more load, then the vehicle will return to the depot before the time deadline of the depot expires. Time windows can be single sided or double sided. In case of a single sided time window, the pickup points usually simply specify the deadlines by which they must be serviced. Vehicles can come early to the nodes without any penalty. However in double sided time window cases, the nodes impose both earliest and latest service time.A vehicle arriving earlier than the earliest service time of a node will incur waiting time. This penalizes the transport management either in direct waiting cost or the increased number of vehicle due to the fact that fewer number of nodes are serviced by a vehicle with a lot of waiting time. Savelsbergh has pointed out that finding a feasible solution to Traveling Salesman Problem with Time Windows (TSPTW) is NP-complete[33]. But with multiple vehicles, the problem becomes NP-Hard. Because of its complexity and usefulness in real life, over the years, VRPTW has been the focus of intensive studies. Surveys about VRPTW can be found in Desrosier et al.[11], Desrochers, Desrosier and Solomon[10], Golden and Assad[17], Solomon[37], etc. Although optimal solution to VRPTW can be obtained using exact method, the computation time is prohibitive for even moderate size problems[10]. Hence heuristic methods are often devised to find optimal or near optimal solutions in reasonable computation time. Several approaches that have appeared in reviewed literature are presented below.

2.2 VRPTW Methodologies

In Marshall Fisher's survey[14], he categorized vehicle routing methods into three generations. The first generation was simple heuristics developed in the 60's and 70's, which were mainly based on local search or sweep. Since these earlier studies were not well documented, it is hard to compare the results they obtained 30 years ago with the more recent solutions. The second generation, mathematical programming based heuristic, were near-optimization algorithms that are very different from normal heuristics. Examples: generalized assignment problems and set partitioning problem to approximate the vehicle routing problem. Their results are usually superior to that of simple heuristics. In fact for linear objective functions, some of these techniques are able to stretch to the optima. The third generation, or the one that is currently undergoing heavy research is exact optimization algorithms and Artificial Intelligence methods. Most successful optimization algorithms are K-tree, Lagrangean Relaxation, etc, while the top AI representatives in VRPTW are Simulated Annealing, Tabu Search and Genetic Algorithms, which are studied by the author in this project. In the following sections, the third generation methods are briefly introduced.

2.2.1 Branch and Bound

Kolen, Rinnooy Kan and Trienekens[25] have presented the first optimization algorithm for VRPTW. The method calculates lower bounds using dynamic programming and state space relaxation. Branching decisions are taken on route-customer allocations. The largest problem solved involved 15 customers.

2.2.2 K-tree Relaxation

Fisher[13] introduces an optimization algorithm in which lower bounds are obtained from a relaxation based on a generalization of spanning trees called K-trees. Capacity constraint are handled by introducing a constraint requiring that some set S, $S \subset C$, of the set of customers must be serviced by at least $k(S)$ vehicles. This constraint is Lagrangean relaxed and the problem resulting is still a K-tree problem with modified arc costs. Time window constraints are treated similarly. A constraint, requiring that not all arcs in a time violating path can be used, is generated and Lagrangean relaxed. The method has solved up to some 100-customer Solomon benchmark problems[37].

2.2.3 Shortest Path Compositions

This is the most successful approach at present. The fundamental observation is, the only constraint[1] that "links" the vehicles together is that each customer in the network must be visited only once. The problem consisting of the rest of the constraints is an Elementary Shortest Path Problem with Time Windows and Capacity Constraints (ESPPTWCC) for each vehicle. Although this problem is strictly NP-hard, very efficient dynamic programming algorithms for the slightly relaxed programs exist. Two decompositions have been investigated computationally: *Dantzig-Wolfe decomposition* and *Variable Splitting*.

Dantzig-Wolfe decomposition exploits the fact that only constraint set (3.3) is linking the vehicles together. The method starts with a solution consisting of a $|C|$ paths, each servicing one customer. The master problem consists of finding a minimum cost set of paths, among all generated paths that together ensures that all customers are serviced. The number of times a path is used is not necessarily an integer, but in the interval [0, 1]. Since the generated paths are not elementary paths, that is the same customer can be serviced more than once on a path, the master problem is the LP-relaxation of a set Partitioning problem. The simplex multipliers of the

[1] See Chapter 3 for the formal constraints of VRPTW.

optimal solution are used to modify costs in the subproblems, which are shortest path problems with capacity and time window constraints. Subproblems are actually identical. If a path with negative marginal cost is found, it is included in the set of paths in the master problem. The method terminates when no negative marginal cost paths can be generated in the subproblem. Now all the paths used in the optimal solution to the master problem have zero marginal cost. If the master problem has integer solution, it is optimal. This method is implemented by Desrochers, Desrosier and Solomon[10], and solved up to some of the 100-customer benchmark problems.

Researchers at Technical University of Denmark[24], on the other hand, suggested using Variable Splitting to solve VRPTW. There are three ways of separating the problem. The model VS_1 consists of a Shortest Path Problem with Time Windows (SPPTW). Model VS_2 consists of a Shortest Path Problem with Time Windows and Capacity Constraints (SPPTWCC). In VS_3 there is a SPPTWCC and a Generalized Assignment Problem (GAP). Olsen has implemented VS_1 method and solved problems with up to 16 customers. Halse has solved up to 100-customer problems with VS_2. VS_3 has yet to be tried.

2.2.4 λ-Interchange Local Search Descent

Thangiah et al.[41] developed a Local Search Descent method that uses a systematic insertion and swapping of customers between routes. A set of such operations are defined as λ-Interchange operators. Due to computation burden, only 1-interchange and 2-interchange are commonly used, in which cases, up to 1 and 2 customers are inserted or swapped at one time. It is a fast algorithm but performs not too well without the help from other heuristics.

2.2.5 Simulated Annealing (SA)

Simulated Annealing searches the neighborhood of *N(S)* in a defined order. The procedure simulates the annealing process in the metallurgy. There is a time associated with the search process. The algorithm jumps to distant location in the search space initially, and makes smaller jumps as time goes on or as it "cools" down. Eventually the process will turn into local search descent method. Osman[30] has applied Simulated Annealing by moving one customer from one route to another or exchanging two customers from two routes.

2.2.6 Tabu Search (TS)

Tabu Search is a memory based search strategy that chooses the best solution contained in *N(S)* that does not violate certain restrictions that prevent cycling. Usually these restrictions are stored as queues in a structure called *Tabu List*. Typical restrictions prevents making a move that has been done within the last t iterations. A solution that has been encountered in the last t iterations is usually forbidden as well. Tabu Search stops after a fixed number of iterations have elapsed. Gerdreau, Hertz and Laporte applied TS using neighborhood that can be constructed by moving a single customer from one route to another. Osman and Talliard [30] used a neighborhood that consists of all solutions obtained from inserting a customer and swapping two customers.

2.2.7 Genetic Algorithm (GA)

Genetic Algorithm codes the VRPTW solutions in forms of bit strings or integer strings, also known as chromosomes in the GA jargon. The method starts with a population of chromosomes. The chromosomes are then paired up and the pair undergoes a crossover and mutation process to produce some children which are different from the parents but inherit certain genetic traits from the parents. This process is continued one generation after another until a fixed number of generations has reached or the population has turned homogeneous or converged. Thangiah has devised a special genetic representation that keeps the polar angle offset in the genes[42]. This method is called Genetic Sectoring Heuristic. The algorithm follows a cluster-first, route-second philosophy and solved 100-customer Solomon problems to near optima. Prinetto, et al.[32] proposed a hybrid GA for Traveling Salesman Problem, in which *2-opt* and *Or-opt* were incorporated with the GA. Blanton Jr. and Wainwright[4] presented two new crossover operators, Merge Cross#1 and Merge Cross#2. Results showed that the new operators are superior to traditional crossover operators.

2.2.8 Constraint Programming

Paul Shaw[35] presented *Large Neighborhood Search (LNS)* to solve VRPTW. Relatedness plays a very important part in the selection of customer to remove and re-insert into the configuration using a constraint-based tree search. Shaw applied *Limited Discrepancy Search* during the tree search to re-insert visits. The results were competitive to those obtained using OR meta-heuristics.

2.2.9 Mixed Heuristics

A number of researchers have attempted to combine some of the heuristics discussed above. In Thangiah's paper[41], he mixed genetic algorithm, SA and TS, and developed GenSat system. The system proved superior to other heuristics at that time in 1993.

Chapter 3

Problem Analysis and Mathematical Model

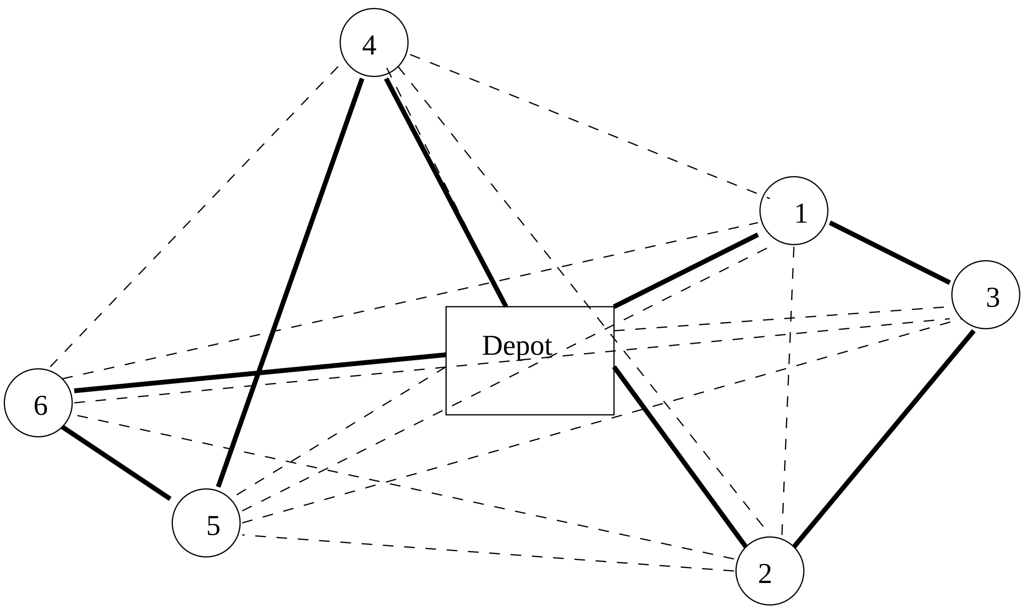


Figure 3.1: A Typical VRPTW Solution

The Vehicle Routing Problem with Time Windows is given by a set of identical vehicles *V*, a special node called the depot, a set of customers *C* to be visited, a directed network connecting the depot and the customers. Let us assume there are *K* vehicles, *V*={*0, 1, 2, ... , K-1*}, and *N+1* customers, *C*={*0, 1, 2, ..., N*}. For simplicity, we denote depot as customer 0, or C_0. Each arc in the network corresponds to a connection between two nodes. It is assumed that any such connection is physically possible. A route is defined as starting from the depot, going through a number of customers and ending at the depot. Number of routes in the traffic network is equal to the number of vehicles used, that is, K. Therefore, exactly K directed arcs leave the depot and K arcs return to the depot. A cost c_{ij} and a travel time t_{ij} are associated with each arc of the network. Every customer in the network must be visited only once by one of the vehicles. Since each vehicle has a limited capacity q_k (identical to all vehicles in our project), and each customer has a varying demand m_i, q_k must be greater than or equal to the summation of all demands on the route traveled by that vehicle k. On the other hand, any customer must be serviced within a pre-defined time interval, limited by an earliest arrival time and latest arrival time. Vehicles arriving later than the latest arrival time are penalized while those arriving earlier than the earliest arrival time incur waiting. Vehicles are also supposed to complete their individual routes within a total route time which is essentially the time window of the depot.

There are two types of decision variables in VRPTW. The decision variable x_{ijk} ($i, j = 0, 1, 2, \ldots, N$; $k = 0, 1, 2, \ldots, K$; $i \neq j$) is 1 if vehicle k travels from node i to node j, and 0 otherwise. The decision variable t_i denotes the time a vehicle starts service at node i. The triangular inequality, i.e. $c_{ij} \leq c_{ih} + c_{hj}$ and $t_{ij} \leq t_{ih} + t_{hj}$, $\forall h, i, j \in N$ need not apply. This is true since we may not be dealing with Euclidean distances here. The objective is to design a set of cost-minimizing routes that service all the customers while all the constraints stated above are satisfied. The model can be mathematically stated as follows:

Parameters: Let

K = total number of vehicles.

N = total number of customers.

C_i = customer i, where $i = 1, 2, \ldots, N$.

C_0 = the depot,

c_{ij} = cost incurred on arc from node i to j.

t_{ij} = travel time between node i and j.

u_i = a fictitious decision variable for node i.

m_i = demand at node i.

q_k = capacity of vehicle k.

t_i = arrival time at node i.

e_i = earliest arrival time at node i.

f_i = service time at node i.

w_i = waiting time at node i.
r_k = maximum route time allowed for vehicle k.

Principle decision variable:

$$x_{ijk} = \begin{cases} 0 & \text{if there is no arc between node } i \text{ and } j; \\ 1 & \text{otherwise.} \end{cases}$$

(VRPTW)

$$Min \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=0}^{K-1} c_{ij} x_{ijk} \tag{3.1}$$

Subject to:

$$\sum_{k=0}^{K-1} \sum_{j=1}^{N} x_{ijk} = K \quad for\ i = 0 \tag{3.2}$$

$$\sum_{k=0}^{K-1} \sum_{j=0,\, j \neq i}^{N} x_{ijk} = 1 \quad for\ i = 1,\ 2,\ ...,\ N \tag{3.3}$$

$$\sum_{i=0,\, i \neq h}^{N} x_{ihk} - \sum_{j=0,\, j \neq h}^{N} x_{hjk} = 0 \quad \forall h \in [1,\ N];\ k \in [0,\ K-1] \tag{3.4}$$

$$u_i - u_j + N x_{ij} \leq N - 1 \quad for\ i = 1,\ 2,\ ...,\ N;\ j = 1,\ 2,\ ...,\ N;\ i \neq j \tag{3.5}$$

$$\sum_{i=0}^{N} m_i \sum_{j=0,\, j \neq i}^{N} x_{ijk} \leq q_k \quad \forall k \in [0,\ K-1]; \tag{3.6}$$

$$\sum_{i=0}^{N} \sum_{j=0,\, j \neq i}^{N} x_{ijk}(t_{ij} + f_i + w_i) \leq r_k \quad \forall k \in [0,\ K-1] \tag{3.7}$$

$$t_0 = 0 \tag{3.8}$$

$$t_i + x_{ijk}(t_{ij} + f_i + w_i) \leq t_j \quad i, j \in [1,\ N];\ i \neq j;\ k \in [0,\ K-1] \tag{3.9}$$

$$e_i \leq t_i \leq l_i \quad k \in [0,\ K-1] \tag{3.10}$$

Constraint (3.2) specifies that there are exactly K routes going out of the depot while (3.3) and (3.4) make sure exactly one vehicle goes into and out of a customer.

Constraint (3.5) ensures that there is no subtours in the solution. (3.6) is the capacity constraint. Maximum travel time for each vehicle is assured in Eq. (3.7). And finally the time window is imposed in Eq. (3.8), Eq. (3.9) and Eq. (3.10).

Chapter 4

Integer Programming Approach

4.1 Overview

Intuitively, the mathematical model of VRPTW included in the Chapter 3 reveals that the nature of the problem can be depicted in terms of a simple integer program, where minimization of total cost is the objective function which is subjected to a number of constraints like trip constraints, load constraints and time window constraints. These constraints forbid the overloading and tardiness in the routes. By integer programs, we mean the decision variables, in our case, whether a specific route from one node to another is taken, can take on only integer values. In fact the decision variables in this program can only be 0 or 1. The output of the integer program is the number of vehicles needed and their complete optimized routes. There are various ways in solving an integer program. One of these is Branch and Bound algorithm. For zero-one problem, a special B&B method called "Additive Algorithm"[39] can be used. The author implemented this algorithm using C++ programming language but the results were not satisfactory. The algorithm turned out to be an *exponentially-bound* algorithm which is very computation expensive as the number of decision variable increases. In fact, with merely a 4-customer problem, the algorithm is practically useless as theoretically it may take weeks to get an optimized solution on an Intel Pentium 133 processor. However for the consistency of the paper this theory and implementation is still briefly included here.

4.2 Comparison of Integer Programming Approaches

Integer Linear programs (ILP) are linear programs which have integer decision variable only. The common Simplex method doesn't apply here because the method involves matrix multiplication and division hence integer solution is not guaranteed. Despite people's constant research of ILP, the computation experience with ILP is far less than satisfactory. To date there does not exist an ILP computer code that can solve the integer programming problem consistently. Currently the most successful algorithms follow three steps.

Step 1. Relax the solution space of the ILP by replacing any binary variable with continuous range $0 \le y \le 1$, and deleting the integer restrictions on all the integer variables. The result of the relaxation is a regular LP.

Step 2. Solve the LP, and identify its continuous optimum.

Step 3. Starting from the continuous optimal point, add special constraints that iteratively modify the LP solution space in a manner that will eventually render an optimum extreme point that satisfy integer constraints.

Two prevailing methods are: *Branch and Bound Algorithm (B&B)* and *Cutting Plane Algorithm.*

Branch and Bound Algorithm This method first uses the integer constraint to branch the LP space into sub-problems. A branch is fathomed if

a) The optimal z-value of LPi cannot yield a better objective value than the current lower bound, or

b) LPi yields better solution than current lower bound, or

c) Current sub-problem has no feasible solution.

If a sub-problem is fathomed, the method switch to another sub-problem on the same level and repeat the process. If not, it will branch this sub-problem further and then go back to the fathoming. Continue this procedure until the optimal solution is obtained.

For example, if we are to solve the following ILP:

$$
\begin{aligned}
&Maxmize\, z = 5x_1 + 4x_2 \\
&\text{Subject to} \\
&x_1 + x_2 \leq 5 \\
&10x_1 + 6x_2 \leq 45 \\
&x_1, x_2 \geq 0 \text{ and integer}
\end{aligned}
$$

The associated LP0 problem has an optimal solution at $x_1 = 3.75$, $x_2 = 1.25$, $z = 23.75$. Obviously both x_1 and x_2 are not integer. So we relax the solution into two branches:

LP1 space = LP0 space + $(x_1 \leq 3)$
LP2 space = LP0 space + $(x_2 \geq 4)$

Figure 4.1 shows how the branching is done graphically. Continue this process until the integer optimum is obtained.

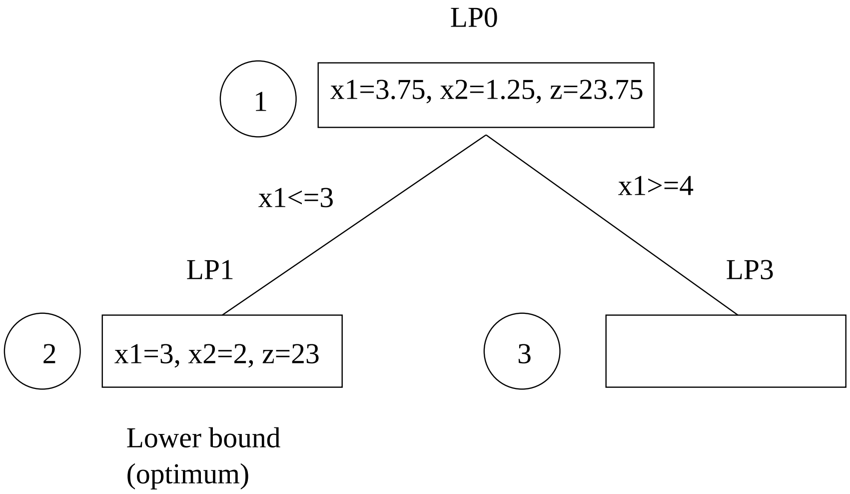


Figure 4.1: An Illustration of Branching

Cutting Plane Algorithm Like in the B&B algorithm, the cutting plane method also started with continuous LP solution. However, rather than using branching and bounding, it modifies the solution space by successively adding specially constructed constraints called cuts. Although Cutting Plane Algorithm is computationally consistent like B&B, it is far less successful. So we are not going into details of this algorithm.

Zero One Implicit Enumeration Algorithm The problem with last two algorithms is both of them involve solving LP using Simplex Method. This is OK with general ILPs, but our problem only deals with 0 and 1 for each decision variable. Going through all the Simplex procedure is obviously an overkill. Nevertheless there is a variation of the general B&B algorithm, called *Zero One Implicit Enumeration Algorithm*. One of the most efficient methods of this type is *Additive Algorithm*. This method enumerates decision variables and branches them between 0 and 1 recursively. It keeps checking for fathoms after branching to delete non-promising branches or sub-problems. The algorithm can be illustrated by the tree structure in Figure 4.2. Because of the simplicity and understandability of the Additive Algorithm, the author decided to implement this algorithm. Next section discusses the revised version of Additive Algorithm. The original Zero-one Theorem was developed by Balas in 1965. In this project, it was revised for our own purpose. The detailed algorithm is included in Appendix B.1. The algorithm initializes the problem by setting all decision variables zero. Decision variables are in terms of an array structure which has a size of $(N+1)^2K$, where N is the number of customers and K is the number of vehicles.

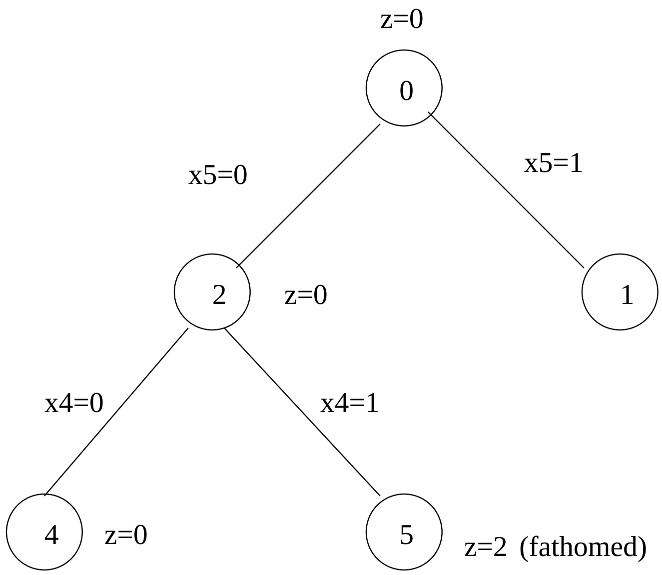


Figure 4.2: A Tree Structure of Zero-one Algorithm

For example, if there are 12 vendors in our problem, $N = 12 + 1 = 13$, $K = 12$, therefore the number of the decision variables is $13 \times 13 \times 12 = 2028$, which is quite a big number!

An upper bound of the cost function was assigned a very large number, in this case, maximal cost of a single trip from one node to another multiplied by number of vendors times two. This is so because the worse case will be when each vehicle goes only to one vendor and comes back to the depot. An array called candidates was initialized to all 1. This global array has the same size of decision variable x, and keeps track of the availability of each decision variable. Then the method goes into branching stage. A variable x_{ijk} with the most slack feasibility was selected and flipped to 1. The arrangement was checked by bound and fathom mechanism. A sub-problem is said to be fathomed if it does no need to be investigated further. Three possibilities will arise:

- If this arrangement is feasible in terms of time and load constraints, it is recorded and the upper bound of cost function is also updated;
- If it is not feasible and cannot be fathomed using all three fathoming criteria, the algorithm loops back to branch function and select the next available best candidate and flip that variable to zero;
- If the arrangement is fathomed, the branch function was terminated and the algorithm returns to a previous configuration i.e. when the chosen decision variable was 0. From there the algorithm will select another variable and branch to 1 again.

The whole algorithm is stopped when there is no more variable to branch and fathom, and the optimal solution is output. The recursive manner of the algorithm was originally done by establishing a push-pop stack. But later author found this was not efficient and incurred some language hazards. Instead recursive C++ functions was used with much better readability and safety.

4.3 Computation Results

A simple main program inputs demand, load and temporal data for 2-vendor problem and outputs best feasible routes. Here is the specification of the test run:

- Maximal route time is 16 time units;
- Capacity of all the track is 6 units;

- Cost of a trip from one node to another is a constant 10 units;
- Demand at vendor 1 is 3 units, at vendor 2 is 4 units;
- Early time is 0 and late time is 8 and service time 1 time unit for every vendor.

It took a Pentium 133 PC less than 1 second to give the following result for this problem set :

```
The Optimal Point is: 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0
For Vehicle 0, The optimal route is 0 -> 2 -> 1 -> 0
For Vehicle 1, The optimal route is 0 -> 0
```

Figure 4.3: IP Results on a Pentium 133

The result shows for this particular day, vehicle 0 goes from the depot to vendor 2 and then vendor 1 and then back to the depot; vehicle 1 stays in the depot. In this sense the program provides an optimal solution of routes as well as minimal number of tracks needed, which, in this case, is one. However when the author duplicate the problem specification for another vendor, i.e. 3-vendor problem, the program takes hours to come up with an optimal solution. The reason for the long computation time is the number of decision variables is equal to $(N+1)^2K$, where N is number of vendors, K is number of vehicles, assumed to be N. If problem involves 3 vendors, there are then $4 \times 4 \times 3 = 48$ decision variables and hence 2^{48} different permutations. This simple number is beyond the capability of most commercial calculator. On the computer the number is evaluated to be almost 2.8E+14. It is estimated that 3320 computer instructions are required to flip one decision variable. If one average instruction takes 3 clock cycles, on a Pentium 133, to flip a decision variable costs

$$\frac{3 \times 3320}{133 \times 10^6} = 7.5 \times 10^{-5} s$$

Even if the Additive Algorithm deletes 99.999% of the sub-problems, it still takes almost 60 hours to give an optimal solution. Later it was found that the computation time can be improved if the slack infeasibilities are not checked each time before a variable could be flipped to 1. In this way the number of instructions goes down to around 460 to decide the value of one variable. The same 2-node problem set but for a two-day schedule was tested again. This time it took 83 seconds to get the optimal solution for this 36-variable problem on a Pentium 133 HP Omnibook. Nonetheless, the B&B algorithm is basically an exponential-factorial time algorithm hence it is still impractical to be applied to VRPTW with large number of nodes.

4.4 Conclusion

In conclusion, Integer Linear Programming problems can be solved using algorithms like Branch and Bound or Cutting Planes, but these algorithms are usually computation expensive and inconsistent with all problem sets. Simple ILP methods are not sufficient to cope with NP-hard VRPTW. Instead, heuristics were further explored in this project.

Chapter 5

λ-Interchange Local Search Descent Method

5.1 Overview

From this chapter onward, we are going to discuss several heuristic methods explored by the author to tackle VRPTWs. Local search algorithms account for a great fraction of all heuristics for VRPTW. Starting from an initial feasible solution, local search procedures use route construction, route improvement or methods that integrate both route construction and route improvement. The first modern heuristic we experimented was λ-Interchange Local Search Descent Method, or λ-Interchange LSD. This method was first introduced by Thangiah, et al. in 1992[40]. An initial routing configuration is obtained through *Push-Forward Insertion Heuristic (PFIH)*. Then a set of insertion and exchange moves are performed on the initial solution in a systematic manner until a pre-defined number of iterations[1] is reached or no improvements can be made. The initial route construction is important because good initial solution tends to be in the closer neighborhood of the optimum in the search space, and hence there is a higher chance of finding the optimum at an earlier stage. We need to define a few more parameters on top of those in Chapter 3.

[1] Iteration is defined as a valid change of one solution state to another thoughout this project.

5.2 Additional Parameters

p_i = polar coordinate angle of customer i, i = *1, 2, ..., N.*
R_k = vehicle route k, k = *1, 2, ..., K.*
O_k = total overload for vehicle k, k = *1, 2, ..., K.*
T_k = total tardiness for vehicle k, k=*1, 2, ..., K.*
D_k = total travel distance for vehicle k, k = *1, 2, ..., K.*
W_k = total travel time for vehicle k, k = *1, 2, ..., K.*
$C(R_k)$ = cost of the route R_k based on a cost function.
C(S) = sum total cost of individual routes $C(R_k)$.
α = weight factor for the total distance traveled by a vehicle.
β = weight factor for the latest arrival time of a customer.
γ = weight factor for the difference in polar coordinate angles.
ϕ = weight factor for the travel total time of a vehicle.
η = penalty weight factor for an overloaded vehicle.
κ = penalty weight factor for the total tardy time in a vehicle route.

It is noted that in our model now and subsequent chapters, t_{ij} is used for the distance between node i and j as well. Hence all distances in this project are measured in terms of time.

5.3 Push-Forward Insertion Heuristic (PFIH)

The Push-Forward Insertion Heuristic was introduced by Solomon[37] in 1987 as a method to create initial route configuration. PFIH is an efficient method to insert customers into new routes.

Let us assume a route $R_p = \{C_1, ..., C_m\}$, where C_1 is the first customer and C_m is the last customer with their earliest arrival and latest arrival time defined as e_1, l_1 and e_m, l_m respectively. The feasibility of inserting a customer into route R_p is checked by inserting the customer between all the edges in the current route and selecting the edge that has the lowest travel cost. For a customer C_i to be inserted between C_0 and C_1, the insertion feasibility is checked by computing (1) the total load, (2) the total travel time, and (3) the amount of time that the arrival time of t_1 is pushed forward. Insertion is only warranted if none of the constraints are violated. The first two can be easily obtained by adding the demand of customer

i to the previous load and adding two route distances (to node i and from node i) plus the wait time and service time to the total time. A change in the arrival time of t_1 could affect the arrival time of all the successor customers of C_1 in the current route. Therefore, the insertion feasibility for C_i needs to be computed by sequentially checking the pushed-forward values of all the successor customers C_j of C_i. The pushed-forward value for a customer C_j is 0 if the time propagated by the predecessor customer of C_j, by insertion of C_i into the route, does not affect the arrival time t_j. The sequential checking for feasibility is continued until the pushed-forward value for a customer is 0 or a customer is pushed into being tardy. In the worst case, all customers are checked for feasibility. The PFIH starts a new route by selecting an initial customer and then inserting customers into the current route until either the capacity of the vehicle is exceeded or it is not feasible to insert another customer into the current route in terms of time. The cost function for inserting a customer C_i as the first node into a new route is as follows:

$$\text{Cost of Inserting } C_i \text{ as First Node} = -\alpha t_{oi} + \beta l_i + \gamma((\frac{|p_i - p_j|}{360})t_{0i}) \tag{5.1}$$

where p_i is the polar angle if the customer in question and p_j is the polar angle of the last visited customer in the last formed route. The unrouted customer with the lowest cost is selected as the first customer to be visited. The polar coordinate angle of the customer with respect to the depot in (5.1) is normalized in terms of the distance. This normalization allows comparison of the distance, latest deadline and angular value of the customer in terms of common unit. Equation (5.1) implies that the desired node to be inserted is the one that is distant from the depot, that has an early time window and that is not too far from the last node that the algorithm has visited, given the parameters are positive. The weights for the three criteria were derived empirically and were set to α = 0.7, β= 0.2, γ= 0.1. Once the first customer is selected for the current route, the heuristic selects from the set of unrouted customers the customer $j*$ which minimizes the total insertion cost between every edge $\{k, l\}$ in the current route without violating the time and capacity constraints. The customer $j*$ is inserted in the least cost position between $\{k* , l* \}$[2] in the current route and the selection process is repeated until no further customers can be inserted. At this stage, a new route is created and the above is repeated until all customers are routed. It is assumed that there is an unlimited number of vehicles available to route the customers, and the actual number of vehicle

[2]Equation (5.3) determines the insertion cost of a normal node as opposed to (5.1).

deployed, K, is determined by the heuristic. The flow of the PFIH is described in Appendix B.2.

5.4 λ-Interchange Local Search Descent Method (LSD)

The effectiveness of any iterative local search method is determined by the efficiency of the generation mechanism and the way the neighborhood is searched. A λ-interchange generation mechanism was introduced by Osman & Christofides[**?**] for the capacitated clustering problem. It is based on customer interchange between sets of vehicle routes and has been successfully implemented with a special data structure to other problems by Osman[30], Thangiah[40], etc. The λ-interchange generation mechanism for the VRPTW is described as follows.

Given a feasible solution for the VRPTW represented by

$$S = \{R_1, ..., R_p, ..., R_q, ..., R_k\}$$

, where R_p is a set of customers serviced by a vehicle route p. A λ-interchange between a pair of routes R_p and R_q is a replacement of subset $S_1 \subseteq R_p$ of size $|S_1| \leq \lambda$ by another subset $S_2 \subseteq R_q$ of size $|S_2| \leq \lambda$, to get the two new route sets $R'_p = (R_p - S_1) \cup S_2$, $R'_q = (R_q - S_2) \cup S_1$ and a new neighboring solution $S' = \{R_1, ..., R'_p, ..., R'_q, ..., R_k\}$. The neighborhood $N_\lambda(S)$ of a given solution S is the set of all neighbors $\{S'\}$ generated by the λ-interchange method for a given λ.

The order in which the neighbors are searched is specified as follows. Let the permutation σ be the order of vehicle indices in a given solution $S = \{R_1, ..., R_p, ..., R_q, ..., R_k\}$ (e.g. $\sigma(p) = p$, $\forall p \in [0, K\text{-}1]$). An ordered search selects all possible combination of pairs (R_p, R_q) according to (5.2) and σ without repetition. A total number $C_{k,2} = \frac{K \times (K-1)}{2}$ of different pair of routes (R_p, R_q) are examined to define a cycle of search in the following order:

$$(R_{\sigma(1)}, R_{\sigma(2)}), ..., (R_{\sigma(1)}, R_{\sigma(K)}), (R_{\sigma(2)}, R_{\sigma(3)}), ..., (R_{\sigma(K-1)}, R_{\sigma(K)}) \qquad (5.2)$$

For heuristics based on a descent algorithm the same permutation σ is used after each cycle of search is completed.

Furthermore for a given pair (R_p, R_q) we must also define the search order for the customers to be exchanged. We consider the case of $\lambda = 1$ and $\lambda = 2$ for the

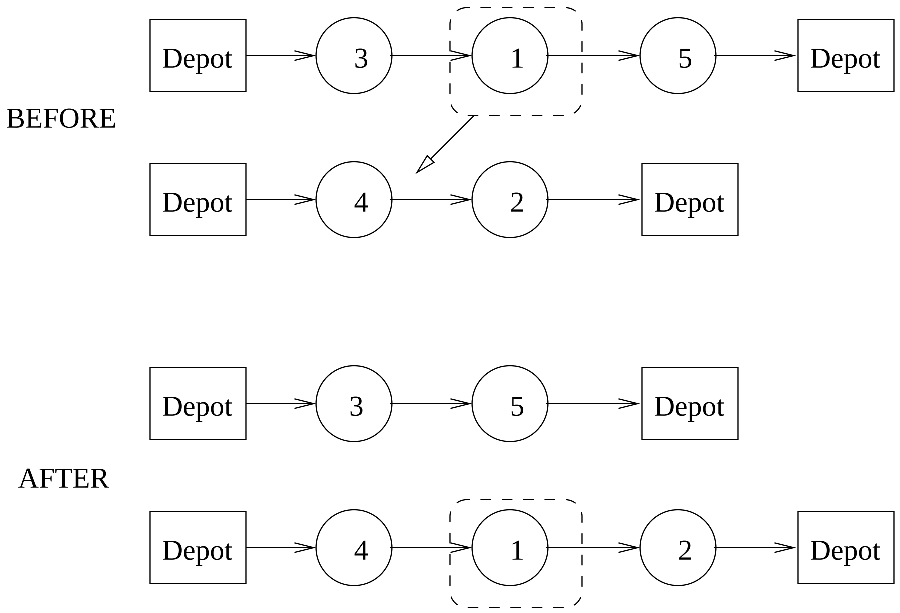


Figure 5.1: An Operation (1, 0)

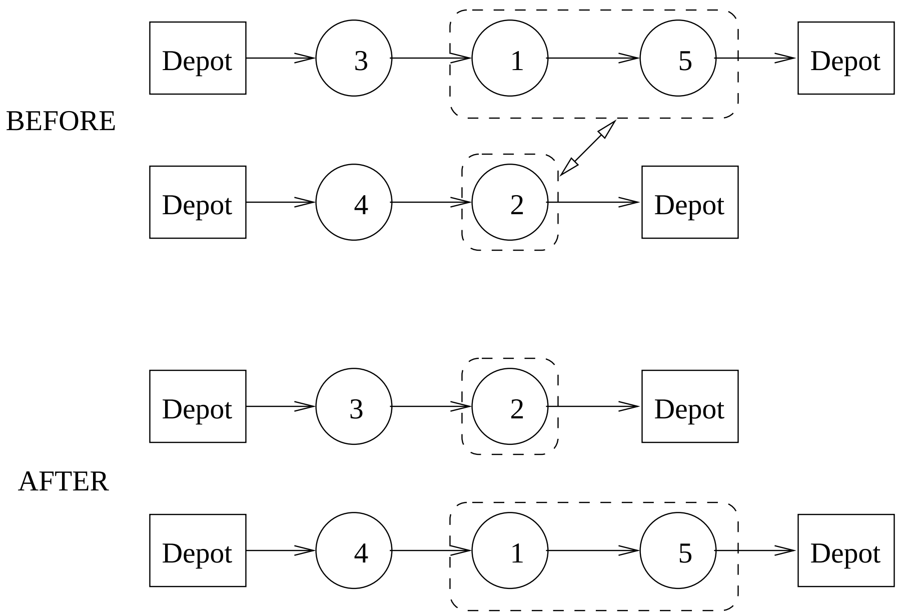


Figure 5.2: An Operation (2, 1)

neighboring search since larger λ involves searching a bigger neighborhood hence the computation intensity is not justified. The λ-interchange method between two routes results in customers either being shifted from one route to another, or customers being exchanged with other customers. The operator (0, 1) on routes (R_p, R_q) indicates a shift of one customer from route q to route p. The operators (1, 0), (2, 0) and (0, 2) indicate shifting of customers between two routes. The operator (1, 1) on routes (R_p, R_q) indicates an exchange of one customer between route p and q. Likewise, the operators (1, 2), (2, 1) and (2, 2) indicate exchange of customers between vehicle routes. Figure 5.1 and Figure 5.2 are examples that illustrate these operations. 1-neighborhood is defined by operations (1, 0) and (0, 1), and 2-neighborhood is defined by all these operations. Note we have defined that (2, 0) shifts two adjacent nodes in the route, and so on. In this way, all these operators have only four operants each: (R_p, i, R_q, j). For example, in operation (2, 0), i is the position of the first of the two nodes in R_p to be removed, and j is the new position of those two nodes in R_q. In operation (2, 1), on the other hand, two nodes starting from position i are removed from R_p and re-inserted into R_q at j, and one node at position j is removed from R_q and re-inserted at i in R_p. This simple definition, which binds two adjacent nodes and move them together, greatly reduces the size of neighoborhood to be searched. One may find that operators (1, 0) and (0,1) are already sufficient to move a solution to another other solution in the search space, but such moves may not be guaranteed with more constraints imposed. The introduction of the rest of the 2-neighborhood operators give rise to greater opportunity of finding a feasible solution without compromising too much computation time. The customers in a given pair of routes are searched sequentially and systematically for improved solutions by the shift and exchange processes. The order of search we implemented for 2-interchange uses the following order of operators (0, 1), (1, 0), (1, 1), (0, 2), (2, 0), (2, 1), (1, 2) and (2, 2) on any given pairs to generate neighbors. After a solution is generated a criterion is required for accepting or rejecting a move. An acceptance criterion may consider many solutions as potential candidates. Two selection strategies are proposed to select between candidate solutions.

1. The FirstBest (FB) strategy will select the first solution in S' in $N_\lambda(S)$ in the neighborhood of S that results in a decrease in cost with respect to a cost function.

2. The GlobalBest (GB) strategy will search all solutions S' in $N_\lambda(S)$ in the

neighborhood of S and select the one which will result in the maximum decrease in cost with respect to a given cost function.

From here one can easily see that the order of search is only important to First-Best search strategy. A move which is a transition from one solution to another in its neighborhood may cause a change in the objective function values measured by $\Delta = C(S') - C(S)$. As the λ-interchange move involves insertion of customers into route, the following cost function is used to compute the cost of inserting customer C_i into route R_k:

$$\text{insertion cost of } C_i = D_k + \phi W_k + \eta O_k + \kappa T_k \tag{5.3}$$

The insertion cost function (5.3) will accept infeasible solutions if the reduction in total distance is high enough to allow either a vehicle to be overloaded or be tardy. Overloading and tardiness in a vehicle route are penalized in the insertion cost function (5.3). The weight factor for total travel time ϕ was set to one percent of the total distance D_k. When calculating the penalty weight factors η and κ in (5.3), η was set to ten percent of D_k and κ to one percent of D_k. The penalty factors were chosen in this manner to allow penalty relative to the total distance traveled by the vehicle.

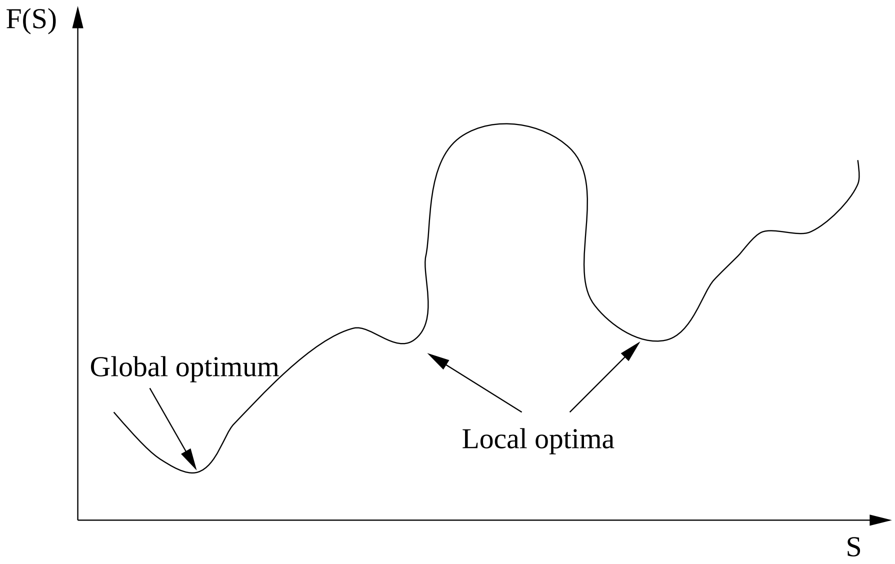


Figure 5.3: A Local Optimum Scenario

In the following we describe the λ-interchange Local Search Descent method (LSD).

LSD starts from an initial feasible solution obtained by the PushForward Insertion Heuristic. The PFIH solution is further improved using the λ-interchange mechanism for a given number of iterations. The procedure of the λ-interchange LSD are shown in Appendix B.3. The LSD result is dependent on the initial feasible solution. In the worst case, LSD is only able to search thoroughly a 2-neighborhood of the initial solution, and if no better moves can be realized the algorithm will exit after a number of iterations. Therefore LSD is doomed to be restricted in a local optimum and unable to explore a larger area as depicted in Figure 5.3. If the initial solution happens to be far from the real optimum, LSD may never minimize the objective function. The LSD uses either FB or GB strategy for selection of neighbors.

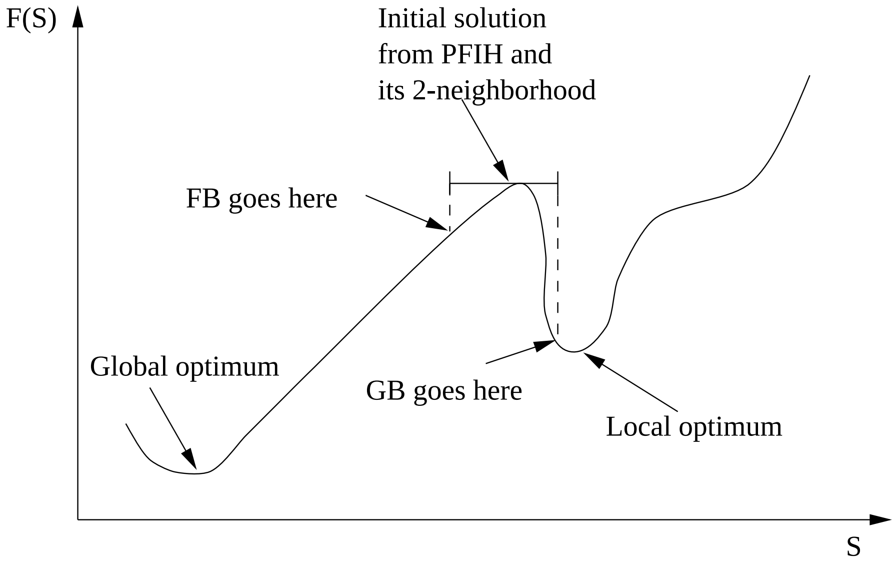


Figure 5.4: A Case where FB is Superior to GB

Global-Best usually achieves better results than First-Best because it keeps track of all the improving moves, but incurs more expensive computation time. On the other hand LSD-FB is a blind search that accepts the first best result. But that is not always the case. Figure 5.4 is such an exception. Following the principle of GB strategy, the algorithm will move to point B after a search of its 2-neighborhood, and will inevitably be trapped in a local optimum. FB strategy, however, may enable the search to jump to point A first and subsequently converge to a real optimum. In this project, we implemented 1-interchange FB as well as 2-interchange GB. Not surprisingly, since FB does not follow a systematic search procedure, it often duplicates solutions and takes longer to complete. And the average results showed that GB is more suitable for LSD. Hence later in other algorithms, when we need to do any local improvements on an obtained solution, we always prefer to use 2-interchange GB strategy. Figure 5.5 illustrates the search path of the objective function value over a number of iterations for a 12-customer benchmark problem K12. The definition of K12 can be found in Appendix A. This problem will also be frequently used in the rest of the report. The result from this run that corresponds to Figure 5.5 is:

The global best solution from 2-interchange is:
After 5 iterations,
Route No. 1 is (80%) 0 (0.0) -> 10 (0.3) -> 9 (1.3) -> 8 (2.1) -> 7 (3.3) -> 11 (4.8) -> 0
Route No. 2 is (100%) 0 (0.0) -> 2 (0.6) -> 3 (1.3) -> 4 (2.3) -> 1 (3.3) -> 5 (4.4) -> 12 (6.0) -> 0
Route No. 3 is (10%) 0 (0.0) -> 6 (0.5) -> 0

The total distance is 9
The total cost is 9
The program has elapsed for 1 secs.

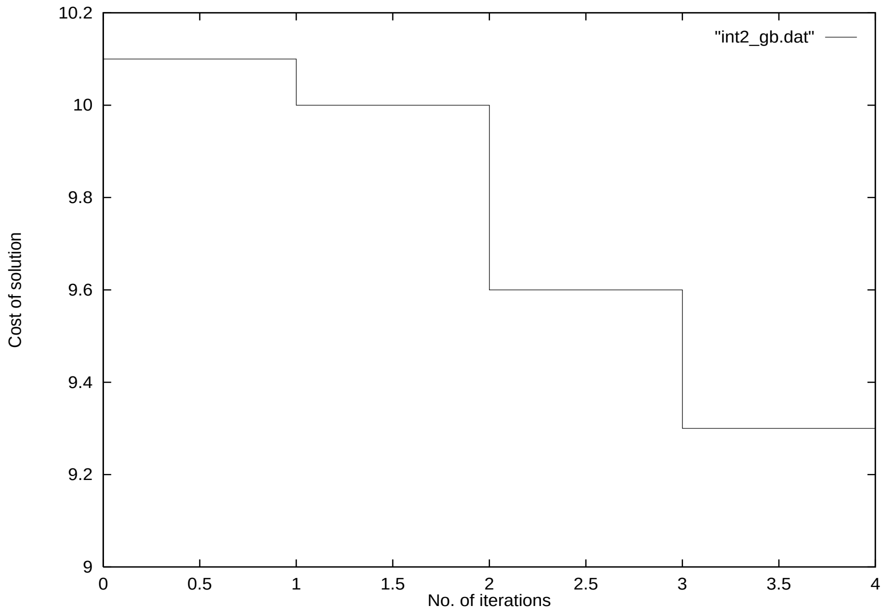


Figure 5.5: Solution Path of 2-interchange GB on K12 Problem

5.5 Summary

To summarize, in solving VRPTW heuristically, the first phase and one of the most important one, is to obtain a reasonably good initial routing configuration. One decent algorithm to achieve this is Push-Forward Insertion Heuristic. Starting from this initial solution, we could apply various local search algorithms to improve the routes. One of the fast and effective heuristic introduced in this chapter is λ-interchange Local Search Descent method. Like every other method, LSD has some disadvantages too. The most prominent one is that the search is largely restricted to a small region thus prone to converge to a local optimum sooner or later. Therefore the quality of a result from λ-interchange LSD heavily relies on the quality of PFIH result. If the real optimal solution is too far from the initial solution, there is very little chance that λ-interchange LSD can find it. Nevertheless we conclude that λ-interchange LSD can be used in coordination with other heuristics to improve solutions locally. Our research later justified such a conclusion.

Chapter 6

Simulated Annealing

6.1 Overview

Simulated Annealing (SA) is a stochastic relaxation technique that finds its origin in statistical mechanics. The Simulated Annealing methodology is analogous to the annealing processing of solids[27]. In order to avoid the meta-stable states produced by quenching, metals are often cooled very slowly, which allows them time to order themselves into stable, structurally strong, low energy configurations. This is called annealing. This analogy can be used in combinatorial optimizations with the states of the solids corresponding to the feasible solution, the energy at each state to the improvement in objective function and the minimum energy being the optimal solution. SA involves a process in which the temperature is gradually reduced during the simulation. Often, the system is first heated and then cooled. Thus, the system is given the opportunity to surmount energetic barriers in a search for conformations with energies lower than the local-minimum energy found by energy minimization. This improved equilibrium can lead to more realistic simulations of dynamics at low temperature. Unlike λ-interchange, SA is a global optimization heuristic based on probability, therefore is able to overcome local optima. Kirkpatrick, et al. are among the first to apply SA to combinatorial optimization problems[23].

6.2 The SA Theory

In the past, Pincus[31] developed an algorithm based on SA analogy to solve discretizations of continuous global optimization problems. Kirkpatrick, Gelatt and Vecchi[23] as well as other people have used SA for discrete combinatorial optimization problems. Formulations of the simulated annealing algorithm for continuous optimization have also been proposed. However most of these formulations, except the last one, lack solid theoretical foundation. Below is a mathematical description of SA.

Assume that we are given a Markov kernel $R(\cdot,\cdot)$, the *selection Markov kernel*, on $(S,\, B_S)$. Here B_S denotes the Borel δ-field on S. Thus for each $x \in S$, $R(s,\cdot)$ is a probability measure on $(S,\, B_S)$. By this we mean that for each n, τ_n is a measurable function from S^{n+1} to $[0,\, \infty]$. The Simulated Annealing algorithm with state space S, with objective function f, with selection Markov kernel R and with cooling schedule τ, constructs a sequence of states $X_0,\, X_1,\, ...,$ a sequence of candidate points $Y_0,\, Y_1,\, ...,$ and a sequence of temperatures $T_0,\, T_1,\, ...$ iteratively, according to the following algorithm:

Simulated Annealing

Step 0. Choose $x_0 \in S$ and $t_0 \in [0,\infty]$. Set n = 0.

Step 1. Select y_{n+1} according to the probability distribution $R(x_n,\cdot)$.

Step 2. Set

$$x_{n+1} = \begin{cases} y_{n+1} & \text{with probability}\, \beta_{t_n}(x_n,\, y_{n+1}) \\ x_n & \text{otherwise} \end{cases} \tag{6.1}$$

and set $t_{n+1} = \tau_{n+1}(x_0,\, ...,\, x_{n+1})$.

Step 3. Increment n and return to Step 1.

In easier language, the SA algorithm can be interpreted as such: At each step of the simulation algorithm, a new state of the system is constructed from the current state by giving a random displacement to a randomly selected particle. If the energy associated with this new state was lower than the energy of current state the displacement was accepted, that is, the new state becomes the current state.

If the new state had an energy higher by d joules, the probability of changing the current state to the new state is

$$\exp(-\frac{d}{kT}) \tag{6.2}$$

, where k is the *Boltzmann constant* and T is the absolute temperature at present. This basic step can be repeated indefinitely. It is known as a *metropolis step*. The procedure is called a *metropolis loop*. It can be shown that this method of generating current states led to a distribution of states in which the probability of a given state with energy e_i to be the current state is

$$\frac{\exp(-e_i/kT)}{\sum_j \exp(-e_j/kT)}. \tag{6.3}$$

This probability function is known as *Boltzmann density*. One of its characteristics is that for very high temperatures, each state has almost equal chances of being the current state. At low temperatures only states with low energies have a high probability of being the current state. These probabilities are derived for a never ending executing of the metropolis loop. Nothing has been said about how many metropolis steps are needed for approximating these probabilities reasonably close. Because experiments have shown that extremely low temperature does not guarantee that the system is in ground state or anywhere near that state, the SA algorithm usually starts from a high temperature and slowly decrease the temperature so that the system is able to search a larger region for the ground state. But how slow is slow? Geman and Geman have shown that if temperature is reduced as $T_k = \frac{T_0}{\log k}$, where T_0 is large enough to allow for initial random configurations, an optimal solution is statistically guaranteed.

6.3 Pros and Cons of Simulated Annealing

The advantages of Simulated annealing are:

- SA can deal with arbitrary systems and cost functions;
- SA statistically guarantees finding an optimal solution;

- SA is relatively easy to code, even for complex problems;
- SA generally gives a "good" solution.

This makes SA an attractive option for optimization problems where other heuristic (specialized or problem specific) methods are not available.

Disadvantages of Simulated Annealing are:

- Repeatedly annealing with a $\frac{1}{\log k}$ schedule is very slow, especially if the cost function is expensive to compute;
- For problems where the energy landscape is smooth, or there are few local minima, SA is an overkill - simpler, faster methods (e.g.,local descent) will work better. But usually one does not know what the energy landscape is;
- Normal heuristic methods, which are problem-specific or take advantage of extra information about the system, will often be better than general methods. But SA is often comparable to heuristics;
- The method cannot tell whether it has found an optimal solution. Some other method (e.g. branch and bound) is required to do this.

6.4 Heuristic Realization of Simulated Annealing

In this project, we successfully realized SA to tackle VRPTW with good results. While inheriting most of the important features of the historical simulated annealing to exploit its advantages in combinatorial optimization problem, the author also made some modifications to the theoretical SA to avoid some of its disadvantages.

Our SA starts with a relatively good solution resulting from PFIH. Initial temperature is set at $T_s = 100$, and is slowly decreased by

$$T_k = \frac{T_{k-1}}{1 + \tau\sqrt{T_{k-1}}} \tag{6.4}$$

, where T_k is the current temperature at iteration k and τ is a small time constant. The square root of T_k is introduced in the denominator to speed up the cool process. Here we use a simple monotonously decreasing function to replace the $1/\log k$

scheme. It is found that this variation produces fairly good results in much less time. The algorithm attempts solutions in the neighborhood of the current solution randomly or systematically and calculates the probability of moving to those solutions according to:

$$P(\text{accepting a move}) = \exp(-\frac{\Delta}{T_k}). \tag{6.5}$$

This is a modified version of Eq. (6.2), where $\Delta = C'(S) - C(S)$, $C(S)$ is the cost of the current solution and $C'(S)$ is the cost of the new solution. If $\Delta \leq 0$, the move is always warranted. One can see that as temperature cools down, the probability of accepting a non-cost-saving move is getting exponentially smaller. However it can also be predicted if Δ is positive and small and T_k is big in the earlier part of the procedure, virtually all moves are granted. When the temperature has gone to the final temperature $T_f = 0.001$ or there is no more feasible moves in the neighborhood, we reset the temperature to

$$T_r = Max(T_r/2,\ T_b) \tag{6.6}$$

, where T_b is the temperature at which the best current solution was found, T_r is the reset temperature, and was originally set to T_s. Final temperature T_f is not set at zero because as T_k decreases to infinitesimally close to zero, there is virtually no probability of accepting a non-improving move, thus the method has been downgraded to LSD. Instead a T_f close to but not equal to zero is more realistic and efficient. To search a local neighborhood, the 2-interchange approach was adopted. And every time a global best solution is found, a 2-interchange (GB) procedure is executed to search for possible elite solutions around it. The procedure terminates after a number of resets.

Depending on the way local neighborhood is selected, we have two implementations of SA. One implementation employs a systematic search of a current solution's 2-neighborhood using (2, 0) and (1, 0) operations, and is called SA-INT. This approach, changing a current solution whenever the acceptance criterion (6.5) is met, is practically a first-best search, therefore may miss out some solution in the space. Moreover, for a neighborhood with very few feasible solutions or the initial solution is close to a local optimum, the method is almost deterministic in terms of the final results. For example, with $\tau = 0.5$ and 3 resets, running SA-INT over K12 problem almost always results in the following solution:

The global best solution from 2-interchange Simulated Annealing is:

After 394 iterations,
Route No. 1 is (80%) 0 (0.0) -> 10 (0.3) -> 9 (1.3) -> 8 (2.1) -> 7 (3.3) -> 11 (4.8) -> 0

Route No. 2 is (100%) 0 (0.0) -> 2 (0.6) -> 3 (1.3) -> 4 (2.3) -> 1 (3.3) -> 5 (4.4) -> 12 (6.0) -> 0

Route No. 3 is (10%) 0 (0.0) -> 6 (0.5) -> 0

The total distance is 9
The total cost is 9
The program has elapsed for 1 secs.

The solution plot of this trial run is shown in Figure 6.1. This solution, compared with the solutions to the same problem by Tabu Search or Genetic Algorithm later, is obviously not optimal. From Figure 6.1, the program finds its final solution just a few iterations after starting, then experiences three temperature resets, prominently indicated by the three impulses. As the temperature goes down to zero, the fluctuation decreases and finally converges to a straight line, which is what we expect. It is noted that the plot has a sharp cut-offs at the bottom, which indicates local optima SA-INT cannot surmount. However, given a complex enough solution landscape, SA-INT can take on probabilistic nature and find some decent results.

Another implementation called SA-RAND, completely relies on randomization in selecting next solution candidates. Since this approach may end up searching a much smaller neighborhood than SA-INT, it is given a $\tau = 0.5$ and larger number of resets to allow more iterations. Result from K12 problem is again shown below and the search path as well as convergence plot are also given in Figure 6.2.

The global best solution from random Simulated Annealing is:

After 2497 iterations,
Route No. 1 is (70%) 0 (0.0) -> 6 (0.5) -> 1 (1.7) -> 12 (2.8) -> 11 (4.4) -> 0

Route No. 2 is (60%) 0 (0.0) -> 3 (0.6) -> 2 (1.7) -> 4 (2.6) -> 5 (3.4) -> 0

Route No. 3 is (60%) 0 (0.0) -> 10 (0.3) -> 9 (1.3) -> 7 (2.6) -> 8 (3.8) -> 0

The total distance is 8.8
The total cost is 8.8
The program has elapsed for 4 secs.

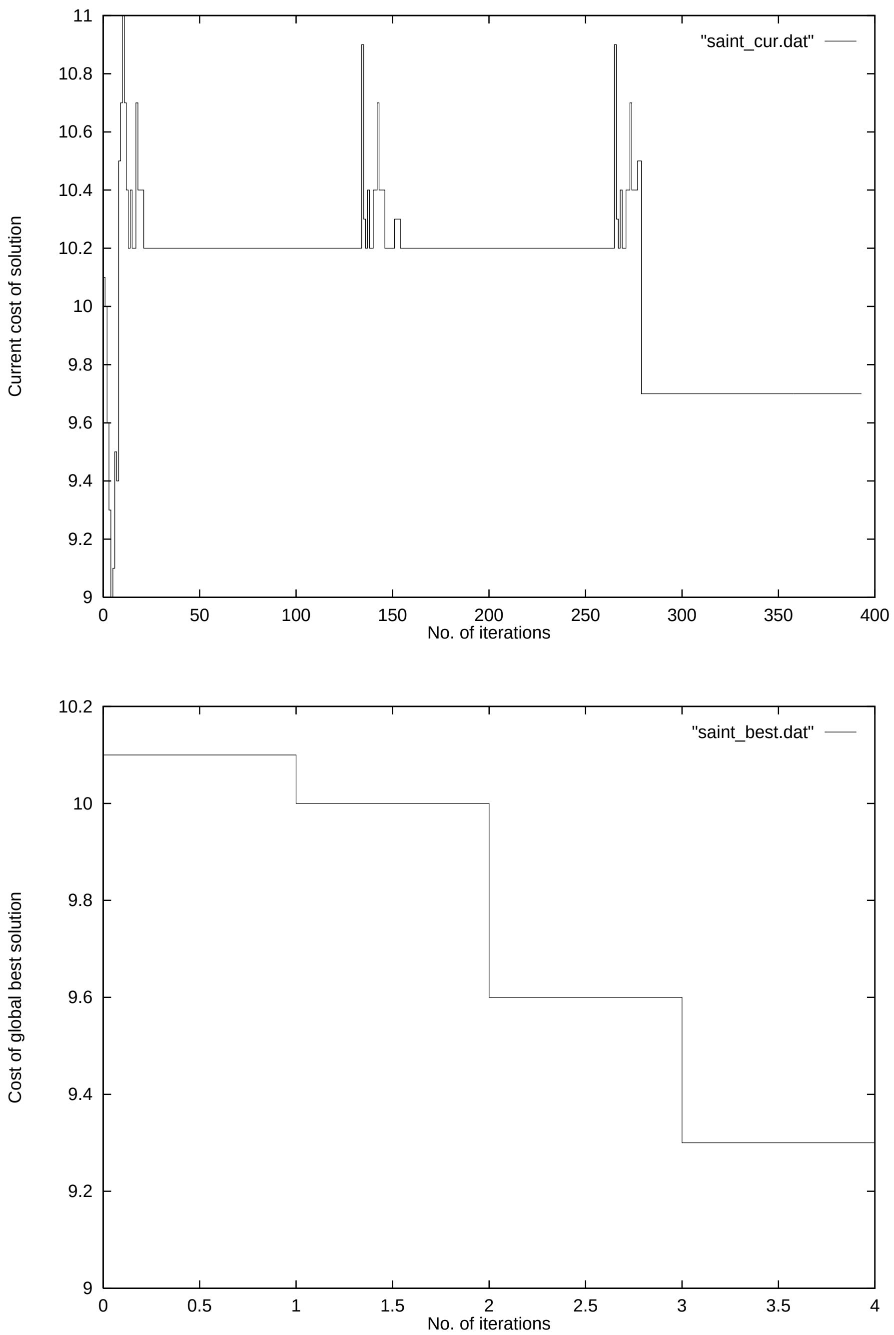


Figure 6.1: Solution Path and Convergence Plot of SA-INT

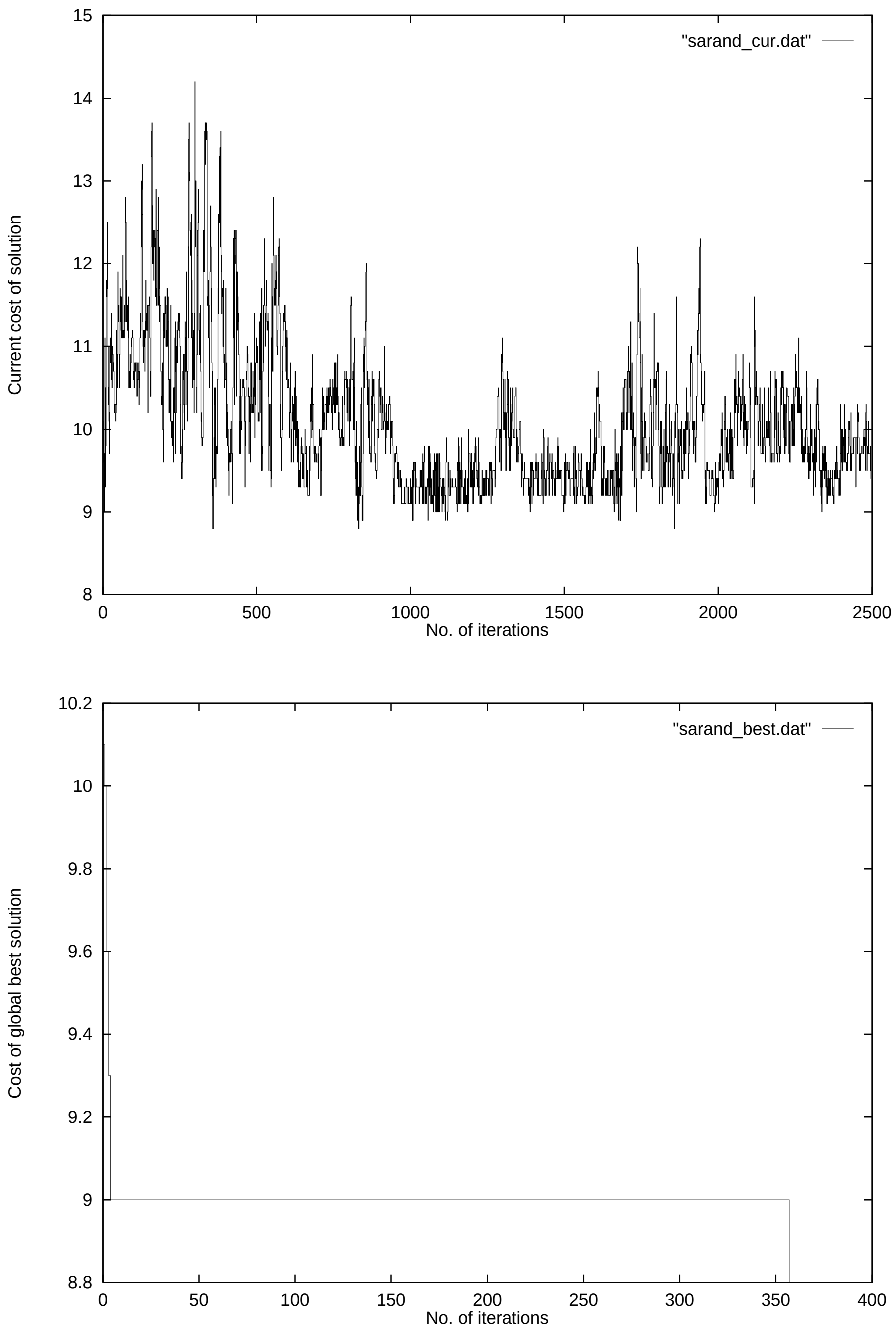


Figure 6.2: Solution Path and Convergence Plot of SA-RAND

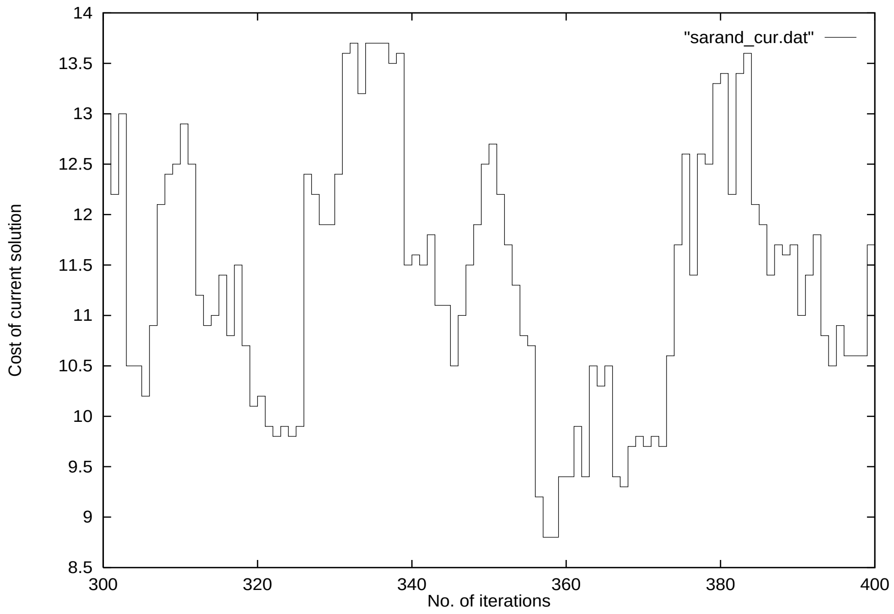


Figure 6.3: An Enlarged View of Figure 6.2

The result we have here is slightly better than that from SA-INT, and was obtained from a different path, as indicated in the solution path plot. Contrary to Figure 6.1 which shows 3 resets and convergences quite clearly, Figure 6.2 does not follow any pattern throughout 2497 iterations. But if we magnify a part of the plot from iteration 300 to 400 in Figure 6.3, we can see that possibly 5 temperature resets were done during these 100 iterations and solution did converge in each cooling process. Generally the "*random walk*" approach of SA-RAND is responsible for the irregular search path. It can be assured that running SA-RAND multiple times can give different final solutions due to its stochastism.

Comparing SA-INT and SA-RAND, the former usually gives better, stable solution for complex problems as it searches a neighborhood more thoroughly but inevitably takes longer time in same number of iterations. However, it is generally believed that given a slow enough cooling schedule, SA-RAND can always find an optimal solution, while SA-INT is more susceptible to local optima. A detailed SA-INT algorithm is included in Appendix B.4.

Chapter 7

Tabu Search

7.1 Introduction

Tabu Search (TS) is founded on ideas proposed by Fred Glover[16]. It is a memory based mata-heuristic method that guides a local search to overcome feasibility barriers. Memory-based strategies are the hallmark of tabu search approaches. The word *tabu* or *taboo* means "banned on grounds of morality or taste or as constituting a risk ..." in the Webster's dictionary. However TS hardly involves references to moral considerations, but rather is concerned with imposing restrictions to guide a search process to negotiate otherwise difficult regions. Restrictions are imposed or created by making references to memory structures which are designed for this specific purpose. TS is based on the premise that problem solving, in order to qualify as intelligent, must incorporate *adaptive memory* and *responsive exploration*. A good analogy is mountain climbing, where the climber must selectively remember the paths he has traveled and must make strategic decision on choice of the way (Figure 7.1). TS achieves this by keeping *Tabu List of* fixed or variable size to record moves and their reverse that have been made or state configurations it has reached, recently or frequently, in the history and mark them as tabu. Sometimes, however, it is favorable to override some tabu moves or states when an *aspiration criteria* is satisfied. Such aspiration criteria could if a move results in a global best, it should be adopted even though it has been made very recently or very frequently.

Together with Simulated Annealing and Genetic Algorithm, Tabu Search was evaluated in the widely referenced report by the Committee on the Next Decade of

Operations Research (CONDOR 1988) to be "extremely promising" for the future treatment of practical applications. Pure and hybrid approaches have already set new record of best solutions in many areas of combinatorial optimization.

Figure 7.1: Adaptive Memory and Responsive Exploration

7.2 Memory and Tabu Search Theory

7.2.1 The Memory

An important distinction in TS arises by the differentiating between short term memory and longer term memory. Each type of memory is accompanied by its special strategies. However the effect of both type of memory many be viewed as modifying the neighborhood *N(s)* of the current solution *s*. The modified neighborhood, denoted by *N*(s)*, is the result of maintaining a selective history of the states encountered during the search. In short term memory approaches, *N*(s)* characteristically is a subset of *N(s)*, and tabu classification serves to identify elements of *N(s)* excluded from *N*(s)*. A TS process that relies strictly on short term memory strategies would allow a solution S to be visited more than once, but it is likely the reduced neighborhood $N^*(s)$ will be different each time around. In long term memory approaches, $N^*(s)$ may be expanded to include solutions not ordinarily found in $N(s)$. With inclusion of such considerations, duplicating a previously visited solution or more generally repeating the visits to a subset of S, is impossible at all. From here, we can see that TS is a dynamic neighborhood method. The neighborhood of a solution s is not static but changes with the history of search. Below we will discuss recency-based and frequency-based TS approaches which are essentially short term and long term strategies, respectively.

7.2.2 Recency-Based Approach

Recency-based memory is a the most commonly used short term memory that keeps track of solutions attribute changes during a recent past.To exploit such a memory, a number of selected attributes of recently visited solutions are marked "*tabu-active*", and solutions that contain tabu-active attributes become *tabu*. This measure prevents the recent past from being include in $N^*(s)$ and hence from being revisited. In addition to exclusion of tabu solution from N*(s), moves that can lead to tabu solutions are also forbidden. However note that this is a short-term memory, therefore solutions visited before the recent past might be revisited as these solutions have exit the tabu list. Nonetheless the neighborhood $N^*(s)$ of such solutions are usually changed.

7.2.3 Frequency-Based Approach

Although in some applications, the short term TS memory are sufficient to produce very high quality results, in general, TS becomes significantly stronger if equipped with longer term memory and its associated strategies. Surprisingly, the benefit of longer term memory does not take a long time to emerge. The improvements can manifest in a reasonable amount of time and allow the search to be terminated somewhat earlier than otherwise possible due to finding a very high quality solution ahead of time. Of course, given longer time, longer term memory is more likely to find better solution, if the optimal solution has yet to be found.

Frequency-based memory provides complementary information to that is provided by recency-based memory, broadening the the foundation for selecting preferred moves. In general, frequences in TS are ratios, whose numerators are the number of iterations where an attribute changes the solutions visited on the particular trajectory (*transition measure*), or the number of iterations where an attribute belongs to solutions visited on a particular trajectory (*residence measure*). Denominators are usually the total occurrence of all the events represented by the numerators. Such ratios produce *transition frequencies* that keep track of how often attributes change, and *residence frequencies* that keep track of how often attributes are members of solutions generated.

A high residence frequency indicates that an attribute is highly attractive if the domain contains the high quality solutions, or conversely, the attribute is extremely unattractive if the domain contains very low quality solutions. On the other hand, a high transition frequency signals an associated attribute is a "crack filler"[16], shifting in and out of solutions to perform a fine tuning function. Therefore in this context, transition frequency may be interpreted as a measure of volatility. Attributes with higher frequency measures, just like with higher recency measures, can trigger a tabu activation rule if they are based on a consecutive solutions that end with the current solution. However these attributes also provide incentives to to determine which moves are selected, as part of the longer term memory strategy.

7.2.4 Intensification

Intensification is a strategy that encourages detailed exploration of the neighborhood of solution configurations or move combinations that are historically found good. A simple intensification approach is included in Figure 7.2. An elite selection strategy is usually to select the solution which has the lowest objective function value from the elite candidate list.

```
Apply TS short term memory.
Apply an elite selection strategy.
do {
      Choose one of the elite solutions.
      Resume short term memory TS from chosen solution.
      Add new solutions to elite list when applicable.
   }
while (iterations < limit and list not empty)
```

Figure 7.2: Simple TS Intensification

7.2.5 Diversification

TS diversification, just like its counterparts in other local search algorithms, aims at driving the search into new regions. Often this is done by modifying the choice rules to bring attributes into the solution that are infrequently used. Alternatively, it can be done by random hops from a frequently visited neighborhood. Such hops are guided by the tabu list so that they do not result in repeated solutions. Diversification strategies are especially useful when optimal solutions cannot be found without overcoming barriers in the search space.

7.2.6 Aspiration Criteria

Aspiration criteria in TS determine when tabu activation rules can be overridden, thus removing tabu classification otherwise applied to a move or a solution configuration. Traditional aspiration criterion consists of removing the tabu classification from a trial move if the move yields a solution better than the best obtained so far. Other criteria, for example those inspired by the concept of *influence* may be effective for improving the search in certain instances. The idea of influence is some moves, though classified as tabu, may be particularly influential under certain circumstances. Such moves should be encouraged by removing the tabu classification.

7.3 Tabu Search for VRPTW

Our design of TS for VRPTW combines short term recency memory and long term frequency memory. We have created a special multi-functional list structure that can serve as a First-In-First-Out queue structure or an array of descending or ascending order.

7.3.1 The Tabu List

When this list structure is used as a queue it is a *Tabu List* that stores two kinds of recency information inside. One is the recently made moves. The moves have such a structure:

$$\langle R_1,\, node1,\, position1,\, R_2,\, node2,\, position2\rangle \tag{7.1}$$

where R_1 and R_2 are two routes under operation, *node1* is a node from R_1 and *position1* is the original position of node1. Likewise for *node2* and *position2*. This notion provides a guideline to avoid making similar moves in the near future. It is noted that such representation does not uniquely describe a move, because full description is very complicated and increase the computation tremendously. Another recency information stored in Tabu List is the solution configurations. Every solution that has been recently encountered is coded into an integer string. For example, if we have the following solution:

Route No. 1 is	0 -> 3 -> 2 -> 4 -> 5 -> 0
Route No. 2 is	0 -> 10 -> 6 -> 1 -> 12 -> 11 -> 0
Route No. 3 is	0 -> 9 -> 8 -> 7 -> 0

The coded integer string is then

$$\langle 3\,2\,4\,5\,0\,10\,6\,1\,12\,11\,0\,9\,8\,7\,0 \rangle\ . \tag{7.2}$$

The zeros are there to delimit the routes. This is, however a representation that has 1-to-1 correspondence to the solution itself. Hence we can convert a solution into such form and backward with ease. Other than the routing information, we also attach the total cost of this solution to the string, for reasons explained in the next section. The lifetime of this tabu moves and tabu solutions on the Tabu List is governed by the *Tabu List Size (TLS)*. The larger the TLS, the longer these moves and solutions remain as tabu. Nevertheless it is found that longer TLS may restrict the search process from proceeding and cause it to end prematurely. Hence in this project, in most occasions, we set the TLS 8 to 10. The Tabu List also employs a frequency measure that counts the number of times each of the tabu moves and solutions have been attempted by the search. This frequency information is important in determining the status of the search process. High tabu-hit frequency implies the search has been caught in a local optimum and hence some action has to be taken.

7.3.2 The Candidate List

The longer term memory approach in our TS algorithm is best represented in the *Candidate List*. This list stores the *elite solutions* the system has discovered in the search process, also in integer string form like (7.2), but ranked according to the total cost attached to the string, instead of a FIFO queue. An item on the Candidate List remains there as long as it has not been intensified. Once in a while, some of these elite solutions may be visited by other paths and become tabu. But since their lifetime as elite candidates are usually longer than that as a tabu, most of these solutions can still be well utilized during intensification. The size of the Candidate List can be variable,and is empirically fixed at twice the size of Tabu List.

7.3.3 The Intensification and Diversification

Our TS procedure begins with an initial solution obtained from PFIH. Assuming that the initial solution is good and close to the ultimate optimal point, it is intensified by undergoing a 2-interchange LSD procedure. One of our implementations, S-TABU, employs a Global-Best approach during intensification. Another implementation known as INT-TABU, uses a First-Best strategy instead. As the current solution is being updated, the moves involved and the solution itself are also being copied to the Tabu List to prevent cycles or duplications. If the current solution is a new global best, it is also copied to the elite list for future exploration. The intensification terminates when the entire $N_2(S)$ has been searched without any improved solutions or the maximum tabu-hit frequency has been reached.

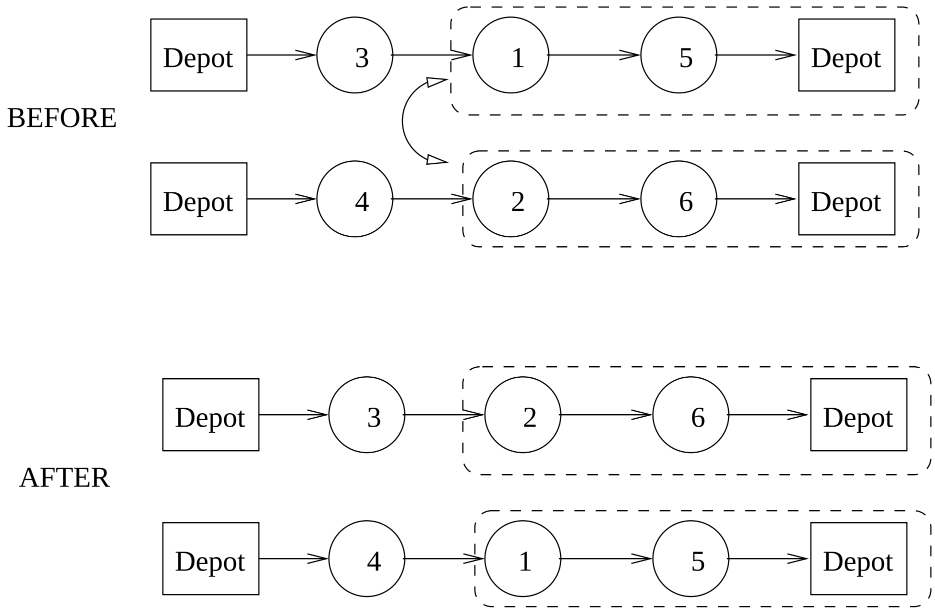


Figure 7.3: A Relink Operation

After one round of intensification, the current solution S is believed to have reached the optimum of its neighborhood, and now it is time to diversify the search to explore other regions. The algorithm accomplishes this by making a series of random 2-interchange hops by the operations (2, 0), (2, 2), (2, 1), and relinking. Relinking is a new operation introduced here to give more flexibility, and is illustrated in Figure 7.3. The hop to a new solution is only granted if this move and resulting new S is not tabu. At all times, a global best solution always overrides the tabu restriction. At every successful random hop, the new solution is recorded in the Candidate List and ranked for further local intensification later. After a number of iterations, the diversification process ends with a simple procedure that selects the least cost solution which is not a tabu from the elite list and updates the current solution with this elite solution. Based on this new solution, a new round of intensification and diversification is triggered. The algorithm stops under one of the following conditions:

- A total number of iteration of the present solution has been reached;
- Unsuccessful diversification as no feasible hops can be made;

To ensure that the global best solution is indeed the optimum in its neighborhood, we apply 2-interchange (GB) at the end of the program. The complete S-TABU algorithm can be found in Appendix B.5.

7.3.4 The Experimental Results and Discussion

The following is one of the better solutions we have found so far for the K12 problem and its corresponding search path graph. This solution is found by S-TABU with maximum number of iteration being 200 and a TLS of 10, 20 hops per diversification process and 200 iterations in each intensification.

The global best solution from S-TABU is:

After 202 iterations,
Route No. 1 is (90%) 0 (0.0) -> 6 (0.5) -> 1 (1.7) -> 5 (2.8) -> 4 (3.8) -> 12 (5.1) -> 0

Route No. 2 is (20%) 0 (0.0) -> 2 (0.6) -> 3 (1.3) -> 0

Route No. 3 is (80%) 0 (0.0) -> 10 (0.3) -> 9 (1.3) -> 8 (2.1) -> 7 (3.3) -> 11 (4.8) -> 0

The total distance is 8.8
The total cost is 8.8
The program has elapsed for 13 secs.

In Figure 7.4, the vertical spikes in the plots indicate the occurrences of diversification and the down-slope steps happened during the intensification. The irregular pattern suggests the S-TABU has successfully surmounted local barriers and explored unknown neighborhood efficiently. The convergence graph essentially proved this point as the trend is moving to better solution if more iterations are granted.

The corresponding solution of this run of INT-TABU is:

The global best solution from INT-TABU is:

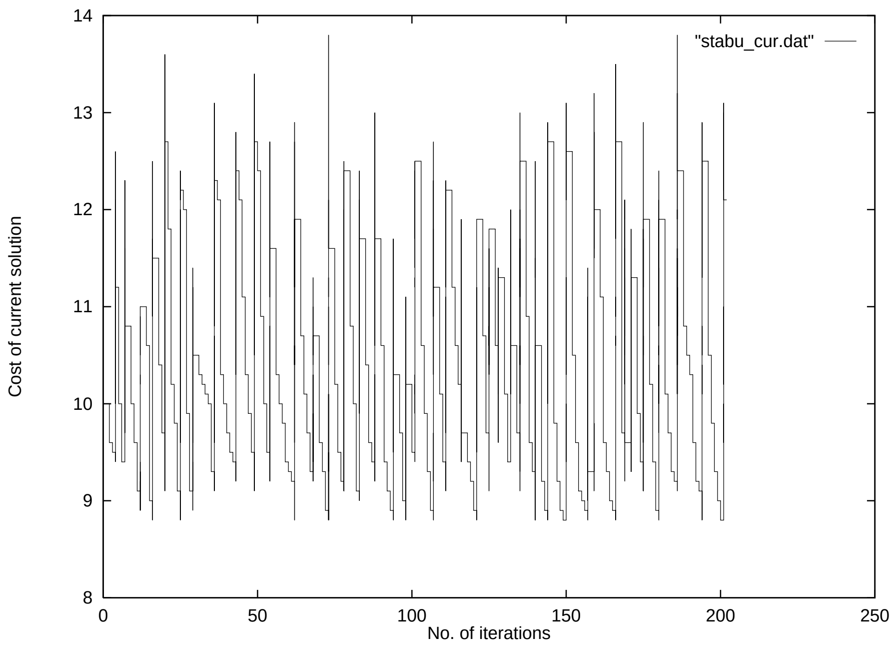
After 514 iterations,
Route No. 1 is (60%) 0 (0.0) -> 10 (0.3) -> 9 (1.3) -> 7 (2.6) -> 8 (3.8) -> 0

Route No. 2 is (70%) 0 (0.0) -> 6 (0.5) -> 1 (1.7) -> 12 (2.8) -> 11 (4.4) -> 0

Route No. 3 is (60%) 0 (0.0) -> 3 (0.6) -> 2 (1.7) -> 4 (2.6) -> 5 (3.4) -> 0

The total distance is 8.8
The total cost is 8.8
The program has elapsed for 5 secs.

Figure 7.5 shows an unpredictable pattern similar to that of S-TABU. However the results of INT-TABU is inferior to S-TABU due to the First-Best approach in its neighborhood search. On the other hand, the author believe, given sufficient number of iterations, INT-TABU can obtain solutions as optimal as S-TABU. In general, TS offers the great flexibility of many parameters, so there is no once-for-all implementation that is suitable for every scenario.



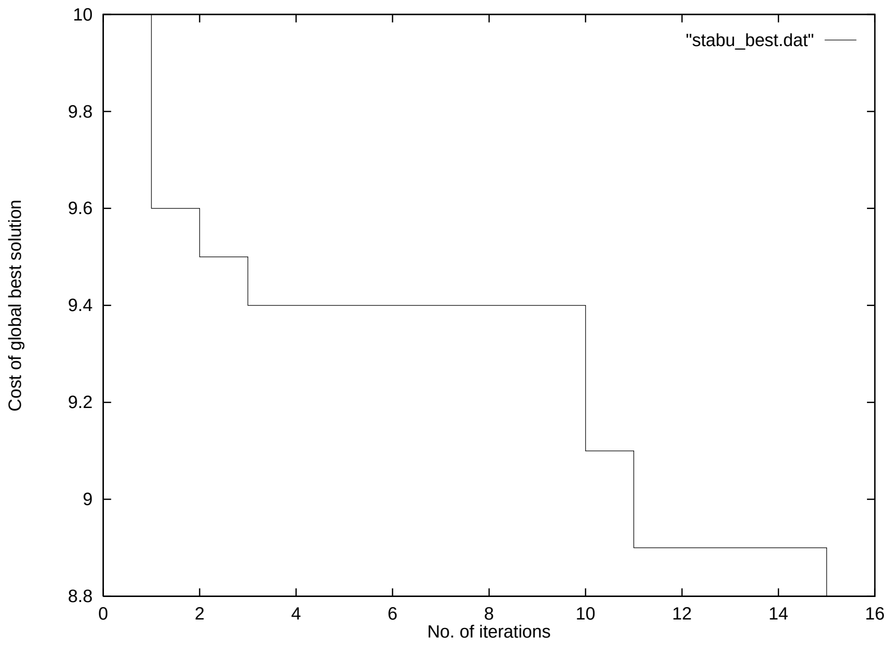
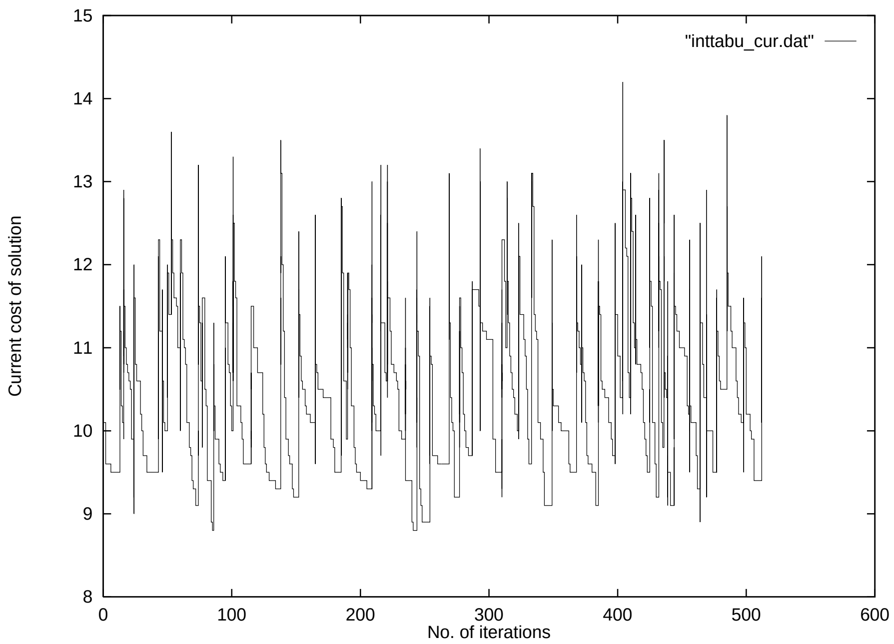


Figure 7.4: Search Path and Convergence Plot of S-TABU



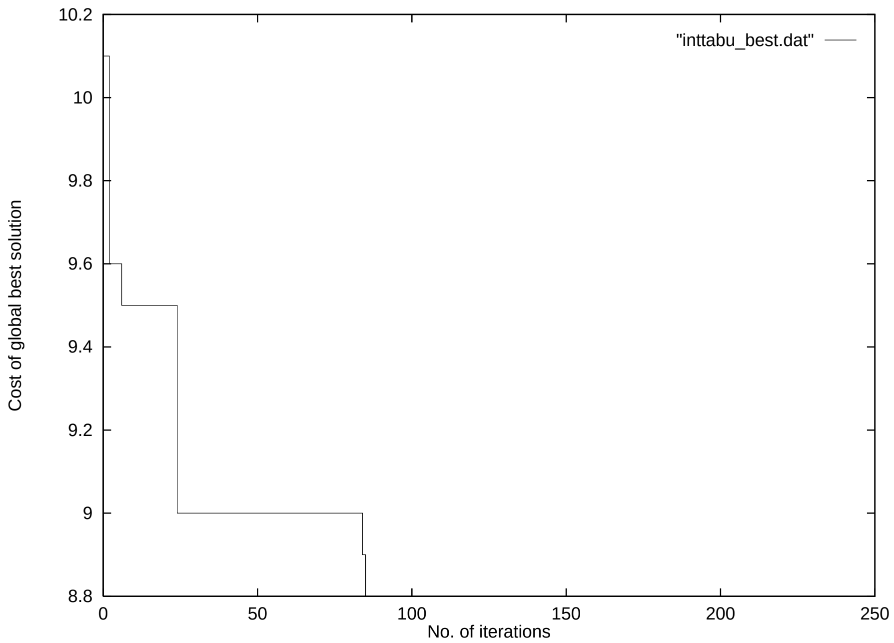


Figure 7.5: Search Path and Convergence Plot of INT-TABU

Chapter 8

Genetic Algorithms

8.1 Background

Genetic algorithm, originally developed by Holland[20], is an adaptive stochastic heuristic that simulates the optimization process with the natural evolution of genes in a population of organics. Whereas most stochastic search methods operate on a single solution to the problem at hand, genetic algorithms operate on a population of solutions over many generations. Joe L. Blanton Jr, et al.[4] first used GA to solve VRPTW in 1993. Thangiah, et al.[42] made use of GA in his "cluster first, route second" GIDEON system in 1995. However generally, GA has not been very successful in the application of VRPTW. Nevertheless the author successfully implemented GA with the help of new crossover operators and other local improvement techniques, and solved 56 Solomon benchmark problems with better results.

8.2 Genetic Algorithm for VRPTW

8.2.1 Introduction

To use a genetic algorithm, one must encode solutions to the problem at hand in a structure that can be stored in the computer. This object is called a chromosome or a genome (Figure 8.1) which comes from a biological context. A chromosome is usually comprised of a fixed number of genes, which in turn, has fixed length, too.

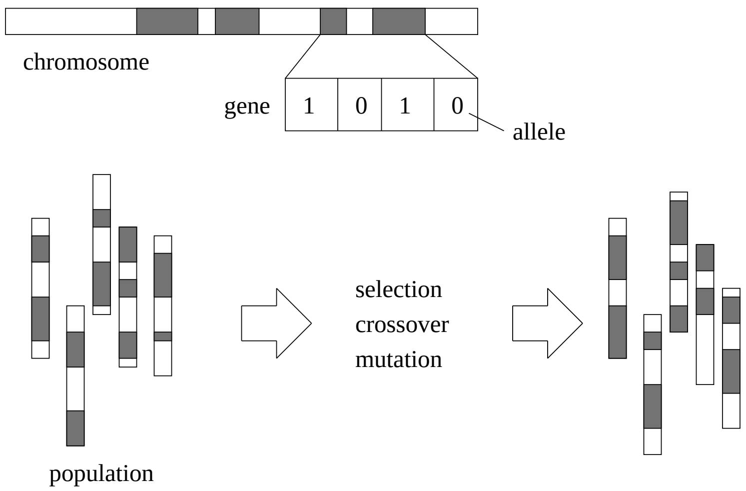


Figure 8.1: The GA Lingos

In case of a binary chromosome like the one in Figure 8.1, the genes are a series of binary digits, each of which is called an allele. However genes can be integers too, which happens to be the case in this project. A fitness value is associated with each chromosome. This value is usually the objective function value of the solution that the chromosome represents, and it measures how "good" the chromosome is. In the VRPTW context, the smaller the fitness the better the chromosome. The GA creates a population of chromosomes then applies crossover and mutation to the individuals in the population to generate new individuals. It uses various selection criteria so that it picks the best individuals for mating (and subsequent crossover). The termination criterion of GA is convergence within a tolerable number of generations.

The genetic algorithm is very simple, yet it performs well on many different types of problems. There are many ways to modify the basic algorithm, and many parameters that can be "tweaked". Basically, if the objective function, the representation and the operators are all right, then variations on the genetic algorithm and its parameters will result in only minor improvements in the overall results.

For any GA, there are five important parameters that determines the performance of its application: representation of solution, initial population, selection, reproduction, and population improvements. We are going though these aspects of our GA implementation for VRPTW in this section.

8.2.2 Chromosome Representation

Like in other GA applications, the members of a population in our GA are string entities of artificial chromosomes. The representation of a solution we use here is an integer string of length N, where N is the number of customers in question. Each gene in the string, or chromosome, is the integer node number assigned to that customer originally. And the sequence of the genes in the chromosome is the order of visiting these customers. Using the same example, the chromosome string that represent the solution in Section 7.3.1 is now:

$$3 \quad 2 \quad 4 \quad 5 \quad 10 \quad 6 \quad 1 \quad 12 \quad 11 \quad 9 \quad 8 \quad 7 \tag{8.1}$$

Note we link the last customer visited in route i with the first customer visited in route $i+1$ to form one string of all the routes involved just like in the Tabu List, but we did not put any bit in the string to indicate the end of a route now, because such delimiters in a chromosome greatly restrain the validity of children produced by crossover operations later. To decode the chromosome into route configurations, we simply insert the gene values into new routes sequentially, similar to PFIH. There is a chance that we may not get back the original routes after decoding, but it is generally assumed that minimizing the number of routes helps in minimizing the total travel cost, therefore packing a route to its maximum capability implies a potential good solution as a result.

8.2.3 Creation of Initial Population

Under the same assumption that we have made in the last chapters, i.e. the solution from PFIH is reasonably good and in the vicinity of the global best solution, we create an initial population related to this solution. The way to do that is by letting the PFIH solution S_0 and its random neighbors $\forall S \in N_\lambda(S_0)$ describe a portion of the starting population. The rest of the population is generated totally on random basis, or unrelated to S_0. The reason for having this mixed population is obvious: A population of members all from the same neighborhood can not go too far from there and hence give up the opportunity of explore other regions. The proportion of relevant chromosomes and random chromosomes are governed by a parameter RAND_RATIO. The higher this ratio, the more diverse the initial population becomes. This parameter also reflects the confidence level of the user to the PFIH solution. If there is a high chance that global optimum is located in $N_\lambda(S_0)$, then it is undoubtedly economical to have a small RAND_RATIO so that the population converges to the optimum sooner.

Other than the composition of a population, the size of the population and number of generations the program is undergoing are also important to the GA performance. From the many experiments, the author finds there is a correlation between the size of the population and number of generations needed to get to the optimum. In a word, larger population requires more generations to reach a global best, and of course takes longer time. Intuitively, this is because a larger population is more diversified and takes more iterations to reach homogeneity. It also has the implication that a moderately small population undergoing more generations is more likely to result in the global best solution. However if the population is too small, GA becomes almost as good as other single solution stochastic methods. In this project we have found that a population of 100 individuals and 500-1000 generations are necessary and sufficient to achieve good results to Solomon's data sets in tolerable time.

8.2.4 Selection

After we have a population of candidates, we need some mechanism to select parents for mating and reproduction. Two types of selection are popular at the moment: *Roulette Wheel Selection*[1] and *Tournament Selection.*

Roulette wheel selection is a complete probabilistic approach. In this method, every chromosome carries a probability of being selected at any time, which is proportional to the normalized reciprocal of its fitness value. In other words, a better solution always has bigger chance of being selected for mating. If we have a population of size N, then a random number generator generates N random number in the range of [0, 1]. Each random number can be treated as a hit to the probability spectrum of the population. The chromosome that has been hit becomes a parent. In this way N parents are thus selected and paired up randomly for further crossovers and mutations. Notice that some of the parents are identical. The apparent drawback of this approach is, if the population constitutes a few good solutions and the a lot more inferior solutions, those good solutions will be selected more frequently than

[1]Roulette wheel selection is also known as natural selection in some literature.

they should be. The result is the population's premature convergence to a local optimum, which may or may not be the global best solution.

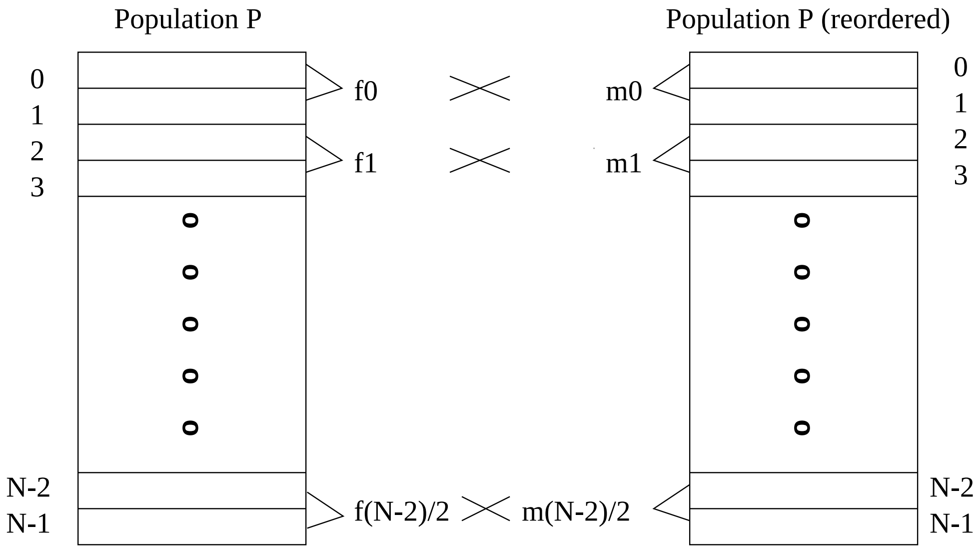


Figure 8.2: The Tournament Selection

Unlike roulette wheel selection, tournament selection does not completely rely on randomization. In this approach, two identical copies of the population of size N are maintained at every generation. In the beginning, both populations are arbitrarily ranked. For population P_1, each pair of adjacent chromosomes (with indices $2i$ and $2i+1$) in the population are compared, the one with smaller fitness value qualifies to be a potential parent and let us call it f_i. After comparing all the pairs in P_1, we have N "fathers", namely f_0, f_1, ..., f_{N-1}. Repeat this process for population P_2, and we get m_0, m_1, ..., m_{N-1}, a set of "mothers" as well. Subsequently f_i and m_i are mated, for all $i = 0, 1, ..., N-1$. The procedure is graphically illustrated in Figure 8.2. The implication in this selection scheme is that genetically superior chromosomes are given priority in mating but average entities have some chances of be selected, too, provided they happen to be compared with a "worse-off" chromosomes.

We implemented both selection methods and results proved our intuitive judgment right. Tournament selection did give better solutions as it promotes a trend toward smaller fitness value without compromising some of the good traits in genetically disadvantaged chromosomes. Therefore we elect tournament selection to be our principle selection method in the GA research.

8.2.5 Reproduction

Reproduction is one of the most crucial functions in the chain of biological evolution. Similarly, reproduction in GA serves the important purpose of combining the useful traits from parent chromosomes and pass them on to the offsprings. It is believed that an efficient and smart reproduction mechanism is largely responsible for high GA performances. The reproduction of GA consists of two kinds of operations, crossover and mutation.

Conventional single/double point crossover operations are relevant to string entities that are orderless, or of different length. They put two integer/binary strings side by side and make a cut point (or two cut points) on both of them. A crossover is then completed by swapping the portions after the cut point (or between two cut points) in both strings (See Figure 8.3). In the context of VRPTW, where each integer gene appears once only in any chromosome, such simple procedure unavoidably produces invalid offsprings that have duplicated genes in one string. To prevent such invalid offsprings from being reproduced, we define a set of order-based crossover operators below.

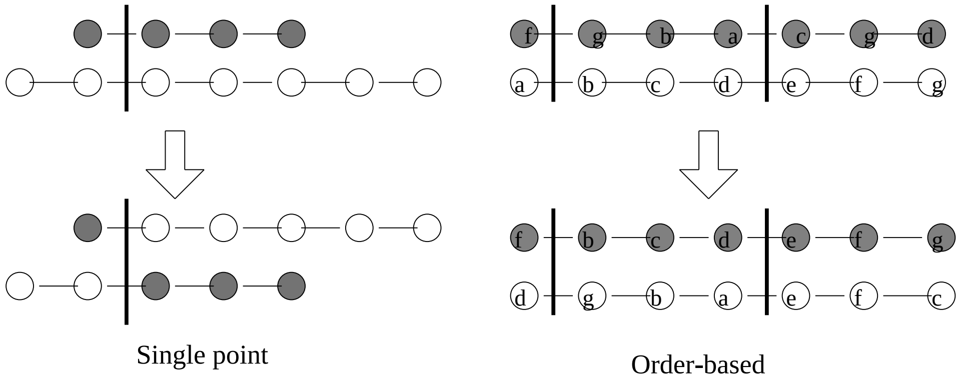


Figure 8.3: Simple Crossover vs. Ordered Crossover

The PMX Crossover[28] The PMX proceeds by choosing two cut points at random:

Parent 1:	$h\ k\ c\ e\ f\ d$	$b\ l\ a$	$i\ g\ j$
Parent 2:	$a\ b\ c\ d\ e\ f$	$g\ h\ i$	$j\ k\ l$

The cut-out section defines a series of swapping operations to be performed on the second parent. In the example case, we swap b with g, l with h and a with i, and end up with the following offspring:

Offspring: $i\ g\ c\ d\ e\ f \quad b\ l\ a \quad j\ k\ h$

Performing similar swapping on the first parent gives the other offspring:

Offspring: $l\ k\ c\ e\ f\ d \quad g\ h\ i \quad a\ b\ j$

Heuristic Crossover A random cut is made on two chromosomes. From the node after the cut point, the shorter between the two edges leaving the node is chosen. The process is continued until all positions in the chromosomes have been considered. Suppose we have the following parents:

Parent 1: $h\ k\ c\ e\ f\ d \quad b\ l\ a\ i\ g\ j$

Parent 2: $a\ b\ c\ d\ e\ f \quad g\ h\ i\ j\ k\ l$

Assume we choose b to be the first gene in the new chromosome, we have to first swap b and g in the the second parents. If the distance $d_{bl} > d_{bh}$, we then choose h to be the next node and swap l and h in the first parent or delete h in the first parent to avoid duplication later. This process is continued until a new chromosome of the same length and comprising all the 12 alphabets are formed. Note that the result varies depending on whether we swapping or deletion is undertaken. We named heuristic crossover with swapping *HeuristicCrossover 1* and *HeuristicCrossover 2* otherwise.

Merge Crossover Unlike heuristic crossover, which rearranges the parents according to distance to produce children, merge crossover operated on the basis of a pre-defined time precedence. This time precedence is often summarized from the time windows imposed by each node. The author created such precedence from the latest arrival time of each node. In the same example:

Parent 1: $h\ k\ c\ e\ f\ d \quad b\ l\ a\ i\ g\ j$

Parent 2: $a\ b\ c\ d\ e\ f \quad g\ h\ i\ j\ k\ l$

Similar to heuristic crossover, a random cut point is selected and a first gene b is chosen randomly from the first parents and swapping is done to second parent. The node which comes earlier in the time precedent becomes the next gene in the new chromosome. Here we define two kinds of merge crossover just like the heuristic crossover case, *MergeCrossover 1* and *MergeCrossover 2*, to differentiate the swapping and deletion schemes.

It is noted that both heuristic crossover and merge crossover produce only one offspring from a pair of parents. Inspired by the fact that both geographic locations and time sequences are important in vehicle routing, we decided to combine the two crossover operators so that two parents now produce two children, each of which comes from either the heuristic crossover or the merge crossover. Naturally we thus have four combined operators from the four combinations, namely *H1M1*, *H1M2*, *H2M1* and *H2M2*. Experimental results indicated that H1M2 outperforms the rest especially in clustered data sets.

Not every pair of parents ought to reproduce in every generation. How many parents are to crossover is governed by the *probability of crossover*, a fix real number between 0 and 1. In our GA, we set it at 0.77, a moderate value usually used in other GA implementation as well. When a couple of parents are determined not to crossover, they are copied verbatim to the next generation.

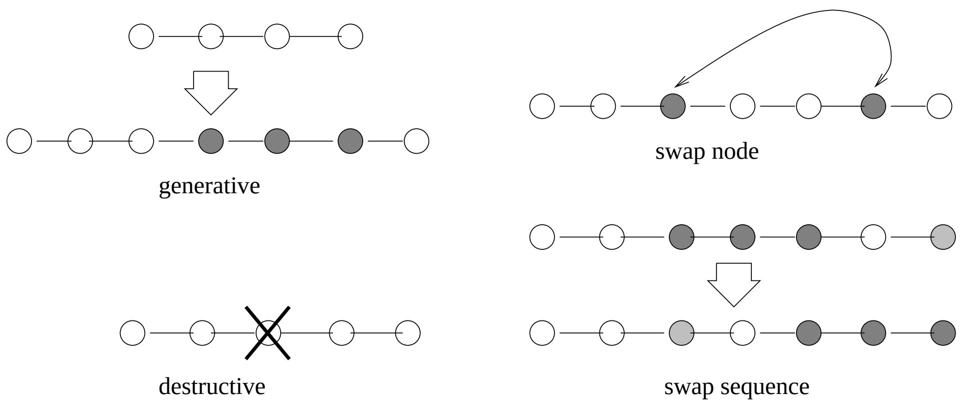


Figure 8.4: Several Common Mutation Operations

Mutation is a complementary operation to crossover. The main purpose of mutation is to avoid over homogeneous population by bringing random, unrelated traits into the present population and increasing the variance of the population. Several mutation operators have been proposed in the literature and they are shown in Figure 8.4. Because the chromosomes are of fixed length in our implementation, only swap node and swap sequence are used here.

There is another important parameter associated with the mutation operations. This is the probability of mutation, or $P_{mutation}$, taking on values from [0, 1]. Earlier research has shown that excessive $P_{mutation}$ values drives the GA into convergence sooner than necessary, often resulting in undesirable local optimal solutions; small $P_{mutation}$ produces the opposite result: intolerably slow convergence. In this project, the author developed a unique *adaptive mutation probability scheme*. The scheme adapts $P_{mutation}$ to the standard deviation of the the population:

$$S = \sqrt{\frac{\sum_{i=0}^{N-1}(x_i - \overline{x})^2}{N-1}} \tag{8.2}$$

where N is the population size, x_i is the fitness value of individual i and $\overline{x}$ is the average fitness of the population. If S is greater or equal to a threshold value MINPOPDEV=5, a minimum $P_{mutation} = P_{min} = 0.06$ is used, otherwise $P_{mutation} = P_{min} + 0.1 \times (MINPOPDEV - S)$. Hence we can see a minimum mutation probability of 0.06 is always guaranteed.

8.2.6 Hill Climbing

Hill climbing is another supplementary measures that takes advantage of local greedy search to improve the chromosomes produced from crossover and mutation procedures. In hill climbing, a portion of the population are randomly selected and decoded into their respective solution form. These solutions then undergo a few iterations of removal and re-insertion operations and are eventually updated with the new, improved solutions. Note this is not a complete λ-interchange procedure due to the moderately large time requirement by each λ-interchange. Furthermore, to reduce the complexity and to prevent from over-reliance on good solutions, the probability of a chromosome to be selected for hill climbing is only set at 0.5.

8.2.7 Recovery of Good Chromosomes

This is yet another ancillary population improving mechanism. In short, the best 4% of a population in the present generation is always retained and used to replaced the worse 4% of the population in the next generation. The rate of recovery has to be chosen with care. The author suggests a rate lower than the probability of mutation because the mutated chromosomes may be totally replaced otherwise.

8.3 Summary and Discussion

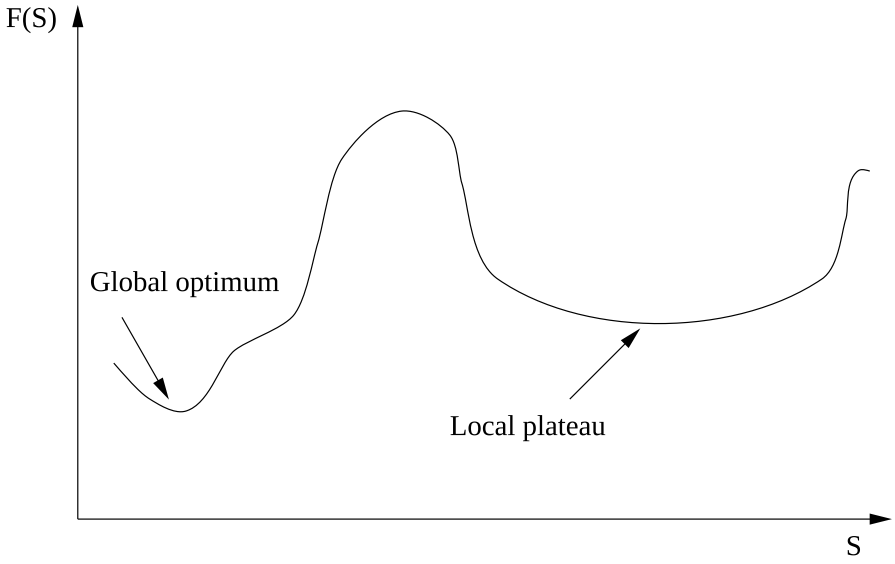


Figure 8.5: A Local Solution Plateau

We tested the roulette wheel selection, tournament selection and the various crossover operators on the K12 problems as well as Solomon's 56 VRPTW instances. The average fitness values against the generation numbers are plotted for four instances of GA on K12 and the graphs are included below.

Comparing the the plots from roulette wheel selection with those from tournament selection, one obvious observation is GA with roulette wheel selection reaches convergence much earlier than GA with tournament selection. This determines the ill-fate of the former when the initial population is not close enough to the optimal solution or the population happens to land on a local optimal "plateau" (Figure 8.5). Next looking at the results from two of the crossover operators we used in this project - *PMX* and *H1M2*, it is not difficult to discover that the *PMX* plots have larger scale zigzags compared to *H1M2*. This can be interpreted as *PMX* is better at introducing instantaneous uncertainties. The reason for this is probably *PMX* is a more general purpose order-based crossover which has no knowledge about the nature of VRPTW, whereas *H1M2* is closely related to time and distance measures.

From the experimental evidences, we believe PMX generally performs well but H1M2 stands out when dealing with either geographically clustered data or time window clustered data, for reason we just stated. For a time, the author attempted to merge these two operators by creating one individual from PMX and one from H1M2 and consequently producing an entire new generation. But the results were unsatisfactory so we still keep them separated at this moment. A skeleton GA pseudo code can be found in Appendix B.6.

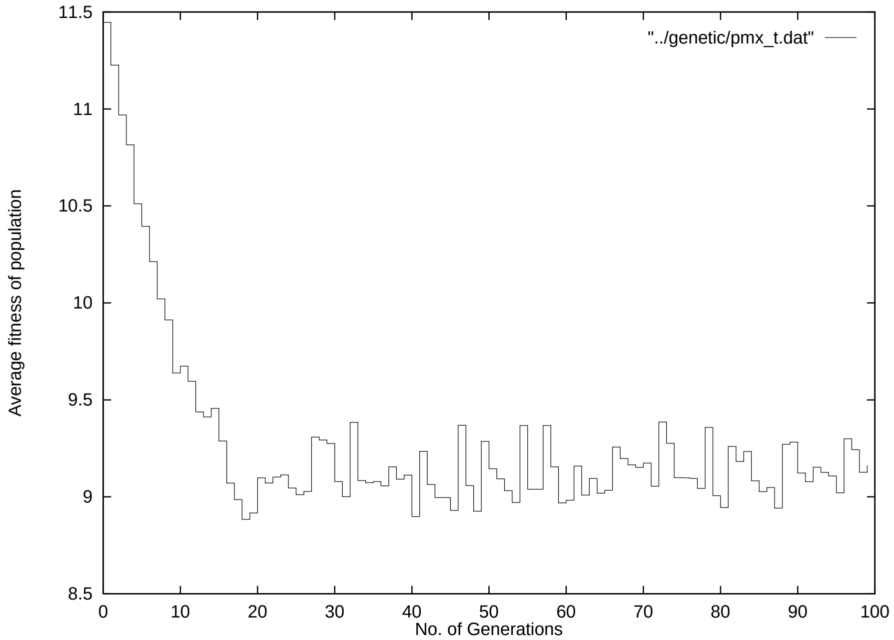


Figure 8.6: Convergence of Tournament Selection with PMX Crossover

When the GA is applied to Solomon's 100-customer benchmark problems, a typical population after a number of generation will look like Figure 8.10.

As a summary, we state the advantages and disadvantages of GA as follows:

Pros

- GA is very flexible with a lot of parameters to adjust for different needs;
- GA generally explores a larger neighbor hood than local search heuristics;
- With proper parameters, GA practices a global optimization that bypasses the local optimum problem;
- Given enough time, GA usually gives good solution.

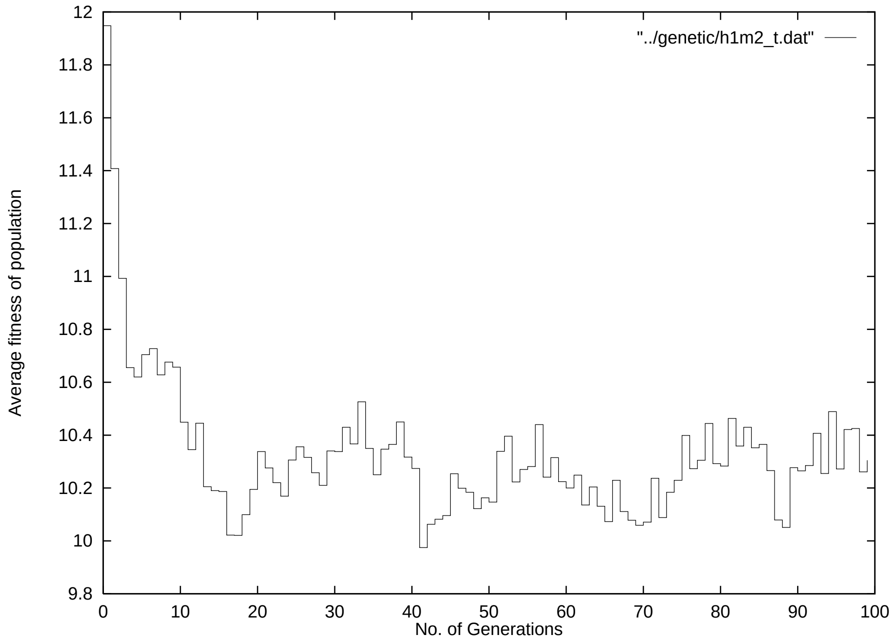


Figure 8.7: Convergence of Tournament Selection with H1M2 Crossover

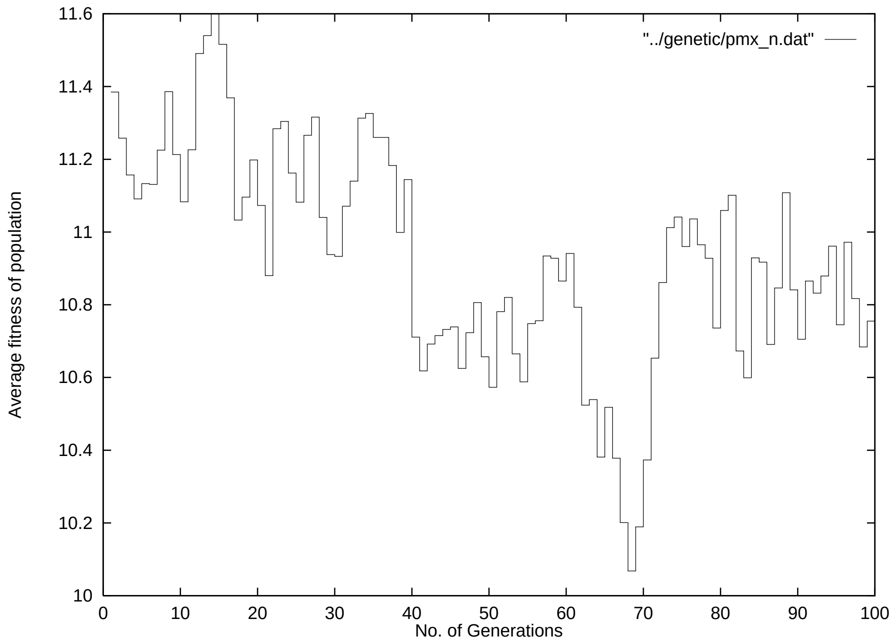


Figure 8.8: Convergence of Roulette Wheel Selection with PMX Crossover

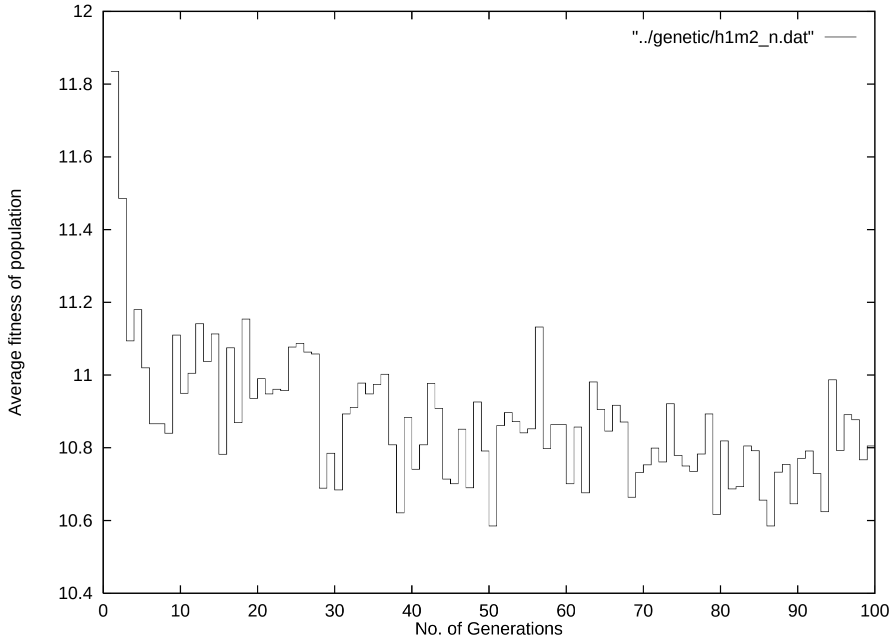


Figure 8.9: Convergence of Roulette Wheel Selection with H1M2 Crossover

At Generation 16, the fitnesses of the population are
2019.66 2023.91 2040.05 2040.05 2052.16 2054.88 2054.88 2054.93 2055.69
2058.042070.75 2070.75 2076.37 2079.36 2089.9 2091.59 2091.95 2091.95 2092.46
2093.722095.04 2095.04 2099.54 2099.87 2099.96 2100.81 2101.27 2101.27 2102.62
2103.032109.88 2111.05 2111.05 2115.67 2115.67 2115.67 2115.67 2118.52 2118.68
2121.962121.96 2123.15 2126.24 2127.59 2128.51 2129.49 2129.87 2131.17 2132.59
2133.182133.68 2134.75 2137.19 2137.69 2137.84 2137.84 2140.36 2140.97 2143.02
2144.062144.84 2146.8 2147.64 2147.65 2150.89 2152.92 2156.23 2156.82 2164.01
2168.672169.67 2170.89 2170.9 2174.56 2175.13 2178.65 2182.64 2187.2 2189.71
2192.692204.27 2206.65 2207.53 2210.5 2217.95 2220.43 2221.76 2222.66 2241.22
2243.242247.19 2252.36 2260.47 2269.17 2272.8 2294.89 2299.65 2308.25 2331.05
2337.69
The standard deviation is 63.3654

Figure 8.10: A Population at Generation 16 for a Solomon VRPTW Problem

Cons

- GA is one of the slowest algorithms in finding the optimum;
- It has no termination criteria other than a number of generations;
- GA can be trapped in a local plateau as the movement of the population is limited by the crossover operations, if that plateau is big and flat enough.

Chapter 9

Results and Comparisons

9.1 Description of Solomon's VRPTW Instances

Each of these 56 instances has 100 customers and the travel time between customers are equal to the corresponding Euclidean distance. The data consist of geographical and temporal differences in addition to differences in demands from customers. Each of the problems in the problem sets has a fleet size of 25, which provides an upper bound for our optimization in terms of number of vehicle K. As these problems vary in fleet size, vehicle capacity, travel time of vehicles, spatial distribution of customers, time window density and width, they are divided into three categories: R-type (uniformly distributed customers), C-type (clustered customers) and RC-type (a mix of R and C types). Two set of problems are proposed for each of these three categories. For example, C103 is one type of data from the C category and C207 is the other type from the same category. Problem sets R1, C1, and RC1 have narrow scheduling horizon, hence fewer customers can be serviced by a vehicle. Problem sets R2, C2, and RC2 have wider scheduling horizon thus more customers can be served on each route. The format of the files are as follows:

The heuristics developed in this project have been applied on all 56 Solomon VRPTW instances and results are discussed in the following sections.

Problem Title						
Vehicle Number	Capacity					
K	Q					
Customer Number	X Coord.	Y Coord.	Demand	Earliest	Latest	Service
0	x_0	y_0	m_0	e_0	l_0	f_0
...	...	...	...	...	...	...
100	x_{100}	y_{100}	m_{100}	e_{100}	l_{100}	f_{100}

Figure 9.1: Solomon's VRPTW Instance File Format

9.2 Comparison Within Our Heurisitics

We conducted most of the tests on a Pentium-II 266 MMX industrial Personal Computer with 32M RAM. Table C.2 lists the best and average results obtained by 2-interchange (FB) and 2-interchange (GB) after up to 20 iterations, as well as times taken to reach the solutions. Accept for problem C109, Global-Best approach invariably yields equal or better results than First-Best in all problems. Whereas in most cases it takes less time to complete for FB approach, GB requires surprisingly little time for problems C101, C105, C109, C201, etc. Each of these problems are actually highly clustered geographically. Therefore their local optima are prominently distinguishable as sharp peaks. In these instances Global-Best search strategy terminates soon after it reaches the apex of the peak.

A comparison of SA-INT and SA-RAND is shown in Table C.3. SA-INT, with $\tau = 0.5$ and 3 resets, completes with better results in almost all 56 problems, and some of the results are already optimal, a pay-off to the longer time it takes to reach a solution. SA-RAND, not much further behind, achieves decent results in much less time of 66 seconds on average, even 200 resets and a lot more iterations. The faster termination can be attributed to the random hit philosophy applied in it. Temperatures are usually resetted before they reach T_f. This implies SA-RAND is free to move randomly most of the time. Another observations is in Table C.3, almost all average solutions are identical to the best solution obtained for SA-INT. In other words, the results of SA-INT are very stable. It proves our argument in Section 6.4 that SA-INT becomes more deterministic than stochastic due to the FB systematic local search approach when increment of total cost is small or zero[1]. A more elegant criterion has to be provided to accept or reject a move. On the other

[1]See Equation 6.5 in Section 6.4.

hand, SA-RAND seldom results in the same solution and is rarely trapped in a local optimum. This is an obvious result of "random walk". However absolute stochastic approach can be so unpredictable that the results are unusable sometimes.

Our implementations of S-TABU and INT-TABU differ in the way local search is done in the intensification process. Here 2-interchange (GB) apparently has an edge over 1-interchange (FB) in INT-TABU. S-TABU yields 16 best known solutions out of the 56 problems it is tested on, whereas INT-TABU obtains only 6 of them. Table C.4 indicates that all best solutions as well as average solutions obtained by S-TABU are more optimal than those obtained by INT-TABU. The disadvantage of S-TABU is the extended computation time. For 500 iterations, S-TABU reaches optima after 3,000 seconds in some of C2, R2 and RC2 problems. There is an implication for the complication of these three types of problems. As we have stated earlier, C2, R2 and RC2 problems have wider scheduling horizon, i.e. there are fewer number of routes in the optimal solution. This situation puts some presure on the quality of the initial solution from PFIH. If the original solution has a lot more routes than the optimal case, it takes a great deal of effort to remove those redundant routes and bring the solution to optimum. Therefore it usually takes longer to solve thess three types of problems. Generally more expensive computation arises as tabu search involves list operations frequently.

Table C.5 shows the results of two GA implementations after 1000 generations with population size of 100. Both GA-PMX and GA-H1M2 performed very well in the clustered data sets, especially GA-H1M2, which optimizes 50% of the C class problems in about 700 seconds. The average costs obtained by GA-H1M2 are also slightly less than those by GA-PMX. The latter did better in the evenly distributed data nevertheless. The author has noticed that both programs did not really converge in most R class and RC class problems, which means given more generations, there is always room for improvement. However due to the constraint in time, we had to stop the program before too long. Another interesting observation is GA's computation burden is almost proportional to the complexity of the problem. For example, in R1 type problems, the time taken by GA-H1M2 is stablized at around 1,400 seconds. This fact can be explained by the termination criterion of GA, which is a fixed number of generations. In each generation, almost fixed number of operations have to be undertaken on the population. GA has essentially encapsulated the uncertainties in the chromsome level, which makes itself tamable for the users.

Putting all these methods together, we can now compare the heuristics we have

Methods Prob.	2-INT (FB)	2-INT (GB)	SA-INT	SA-RAND	S-TABU	INT-TABU	GA-PMX	GA-H1M2
C1	983	965	943	972	874	944	882	872
	31	25	84	33	557	417	544	556
C2	811	780	766	781	644	793	660	641
	37	55	166	71	1885	174	971	1073
R1	1529	1469	1422	1471	1292	1408	1333	1396
	31	46	78	48	1076	578	507	495
R2	1402	1330	1279	1331	1097	1189	1124	1181
	43	98	217	103	3323	1787	851	771
RC1	1774	1680	1657	1680	1471	1584	1547	1619
	31	49	63	44	1016	487	432	485
RC2	1778	1700	1642	1701	1331	1397	1343	1392
	37	60	146	65	3217	1561	835	635

Table 9.1: A Snap Comparison of the Heuristics

Note: the number on top of each cell is the average total cost; the number on the bottom is the average CPU time in seconds associated with the type of problem.

studied in Table 9.1. This table shows the average total cost obtained and the average time taken for each heuristic to solve each category of the problems. We have discovered the following: In general TS is the most effective heuristic, solving the largest number of benchmark problems to optima and achieving the least average total cost in almost all categories. However the computation time is about 2-3 times that of GA and almost 20 times that of SA. GA almost hit as many optima as TS but did not as well as TS in R1 and R2 problems. Nevertheless GA still remains a strong competitor to TS as it is a good compromise of quality and time. SA is very fast and offers reasonably good solutions. All three heuristics tend to give better results to clustered data. The correlation between the number of vehicles required and the total cost incurred in a solution is not straightforward. Fewer number of routes can incur more cost, especially in homogeneous data (R class problems). R109, R201, R202 and R203 are examples of such instances. Detailed comparison of these heuristics can be found in Table C.1.

9.3 Comparison with Best Known Solutions in the Literature

Table 9.2 gives us a clue of how good our heuristics are compared to the best known results in the literature. Average costs of 6 classes of Solomon problems from the

Prob.	Pub. Best	2-INT	SA	TS	GA
C1	827.54	+15.9%	+13.6%	+4.3%	+3.9%
C2	589.4	+32.3%	+28.5%	+5.6%	+4.7%
R1	1191.6	+23.2%	+19.2%	+5.8%	+9.9%
R2	1024.7	+29.5%	+24.8%	+3.3%	+6.1%
RC1	1361.3	+23.2%	+21.1%	+6%	+10.7%
RC2	1156.8	+47.0%	+41.9%	+11.8%	+10.9%

Table 9.2: Relative Average Cost for Our Heuristics Against the Best Known
Note: This table compares the minimal costs between the published best solutions and our solutions. The percentages are derived from Table C.1.

published best solutions are given in column 2 of the table; the average minimal costs generated from four of our methods are presented as percentage increments against the published best. The larger the percentage, the less sucessful the method is. From here we can easily see that λ-interchange and SA alone are in general not suffcient to solve VRPTW to optimal. TS and GA, once again, demonstrate they competitiveness to any other algorithms in all categories but RC2 problems. Table C.1 gives the detailed tally between our best results and the best known results in the literature of all 56 Solomon's problems. In this project we have altogether obtained 18 new best known solutions. Best known results of C class problems are most likely already optimal, hence our solutions tied with them in most cases. In R and RC problems, we obtained a few new best solutions, but our solutions usually require more vehicles than the solutions in history. Nevertheless this is not our concern as we are not optimizing the number of vehicles in this project. It is noted that the best known results could have been obtained from various algorithms and different parameter settings. The source from which we obtained the best known solution did not provide us with the time taken to reach these optima, hence no comparison of computation time can be made.

Chapter 10

Conclusion

10.1 Findings and Contributions

In this final year project, the author first compiled a detailed literature review of optimization methods for VRPTW in the past and at present. This review categorizes the methods by their respective approaches, e.g. simple heuristics, exact methods, and artificial intelligence, etc. This study offers a good guidance for future study of the subject as it informs us where is undiscovered promising area and where is already fully exploited. As the computing power is being enhanced almost every day, we believe that a consistent exact optimization algorithm for solving VRPTW is no longer a miracle.

The implementations of λ-interchange based on Thangiah's theory is an indispensable beginning of our heuristic endeavor, because λ-interchange is a basic corner stone of all the more complex heuristic algorithms later. It clearly defines the meaning of λ-neighborhood and the operators to explore such a neighborhood. The method is simple and straightforward but useful in almost all kinds of local search procedures. Thangiah has never been able to define clearly the operation (2, 0) and (2, 1) in his papers. And these points are clarified in Section 5.4 with figures and literal description. The author's new definition is complete and easy to realize. Theoretically a sequence of 2-interchange operations are able to bring the current solution to anywhere in the solution space.

We next studied Simulated Annealing and Tabu Search, two of the most talked-about combinatorial optimization strategies in the 90's. The author was enthralled by the simplicity of SA and the elegance of TS. One of our two implementations of SA takes a more systematic approach in local search and the other a more stochastic approach, even though both involve random parameters. The former, or SA-INT, is fast enough and outperforms its counterpart SA-RAND. This once again proves the importance of systematic search of neighborhood in local searches. Our strict tabu search (S-TABU) algorithm is one of the many ways to employ the sophisticated memory-based meta-heuristic Tabu Search (TS). S-TABU was able to solve many of the Solomon problem sets to optima in an average of 1500 seconds on a Pentium-II 266 MMX PC. This apparently benefited from the use of longer term memory and diversification strategy. Many authors have implemented their versions of TS as well but none of them in the literature produced as many best solutions as we did in this project. However, our S-TABU still lacks some of the important features of TS, such as influence, and is very buggy due to limited time spent on it. The possibility of further improvement is almost certain.

The main contribution of the final part of the project is a new application of GA to VRPTW. Previously, Thangiah used cluster-first-route-second method when applying GA to VRPTW. In his GIDEON system, the angular differences were coded in the chromosomes. GA was only used to sector the customers within clusters. Other heuristics like 2-opt and simulated annealing had to be used to assist GA in routing the customers within one cluster. Strictly speaking, it is only a hybrid heuristic that constitutes some GA element. The author's current implementation, however, is a complete GA with a more intuitive representation, a complete set of improvement schemes of many flexible parameters. The results from this GA system prove superior than that from the GIDEON system in all 56 Solomon instances. Other authors, like Blanton Jr., et al., applied GA with similar representation to VRPs, but almost all their works were concentrated on Traveling Salesman Problem, which involves only one route and has been proved NP-complete. Moreover, these authors failed to present result data so no comparison can be made. The author has reason to believe that his research is among the first to solve complex VRPTW by GA and to present comprehensive results to all 56 100-customer benchmark problems.

10.2 Applications

The immediate application of our research would be a transport scheduling system for a logistic department of a private company. The physical distribution group of the department usually keeps records of the vendors that they deal with and their locations, demands, travel costs, time windows, etc. Loading these data into the system and run the program on a daily basis could produce the supply/distribution schedule and routing plan for the next day. One can also schedule a weekly or even monthly routing scheme for the vendors.

An more exciting application will be a dynamic on-line transport control system. The routing algorithms are installed on a central mainframe or workstation in the manufacturer center. It has an encrypted interface on the World Wide Web that allows vendors and trucking companies to query or upload information to the system on a real time basis. Based on the information gathered, the computer calculates new routing plans at a short interval and supplies this routing information to on-going trucks through mobile network and in-vehicle display units. If a certain vendor cancels his supply or demand, a change in routing will be reflected accordingly and shown to the vehicle driver almost instantly. He or she can then adjust the routes and visit another vendor. On top of this application, if the vehicles involved are equipped with Global Positioning System, their location, speed, and possible road congestion information could be transfered back to the central control. The algorithm will then take these additional data into concern and reduce the cost even further.

Other related application would be school bus or private bus routing, job scheduling, etc. Furthermore, since the algorithms we have developed are generic, they can be used for other optimization problems after modifications.

10.3 Future Prospects

Presently, we are looking into expanding the heuristics we have developed so far to multi-depot vehicle routing problem. A multi-depot VRP involves more than one depot, which reflects the scenarios of the distribution system of large corporations. These companies have a number of manufacturing operations and need a flow of components from various suppliers to the operations. In such a case, vehicles start from any one depot, visit the vendors and deliver the goods and services to any depots. Naturally this is a more complex model but maneuverable based on the algorithms we already have. In the future we would also try combining some of the heuristics with mathematic programming techniques and solve for exact optimal solutions for VRPTWs.

Appendix A

A Benchmark 12-customer Problem

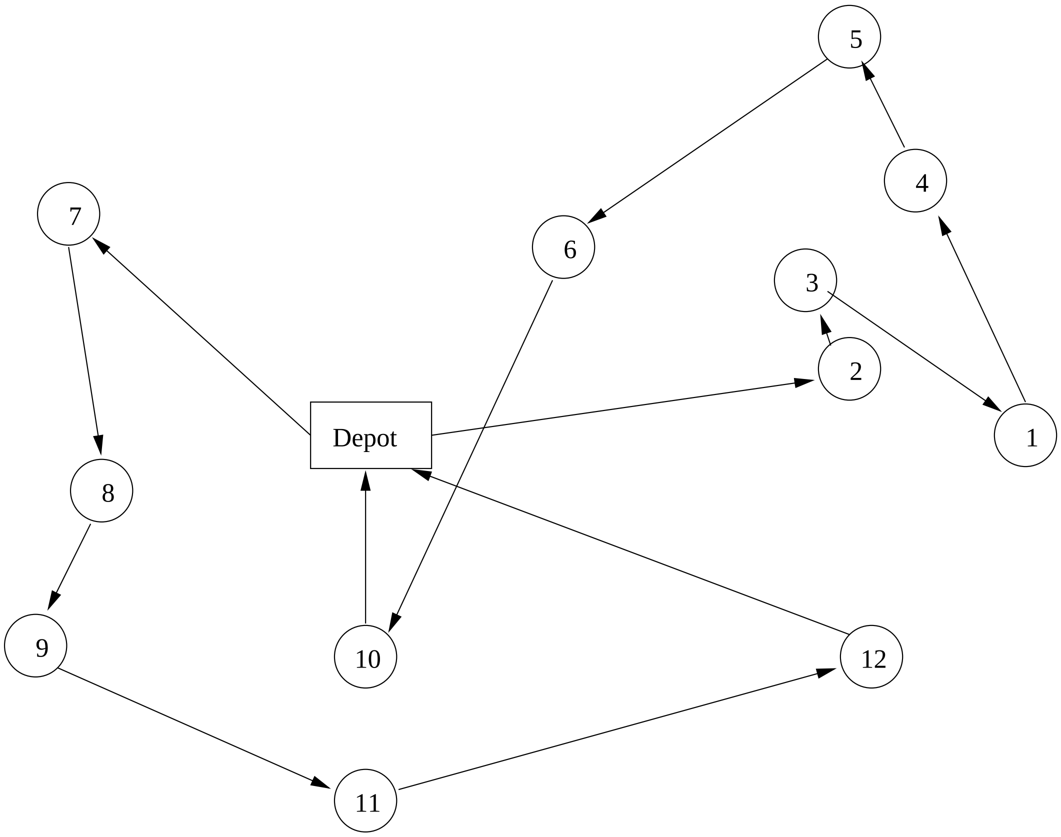


Figure A.1: K12 Problem and a Sample Solution

This 12-customer benchmark problem, code-named K12, is designed by the author during his industrial attachment at Hewlett Packard. It is a simple, un-clustered VRPTW problem set that is suitable for quick testing of algorithms, and it has been used intensively throughout the project. In fact, it has been built into all the executables as a default test problem if no arguments are given to these executable programs. Here is a description of the K12 problem.

K12 depicts a scenario where there is one depot and 12 customers which supply components to the depot. A schematic of the locations of the depot and customers are shown in Figure A.1. The polar angles, Euclidean distances, demands, time windows and service times are tabulated and shown in Table A.1. The depot is marked as node 0. The capacity of each vehicle is 10 units. The demands in the table all come in units.

Nodes	0	1	2	3	4	5	6	7	8	9	10	11	12
Dist.	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1.2	0.6	0.6	1.1	1.2	0.5	1.0	0.5	0.6	0.3	0.6	1.1
1	1.2	0	0.7	0.7	0.5	0.6	0.7	1.6	1.4	1.5	1.2	1.4	0.6
2	0.6	0.7	0	0.2	0.4	0.6	0.5	1.2	1.0	1.2	0.8	1.0	0.6
3	0.6	0.7	0.2	0	0.5	0.6	0.5	1.2	1.0	1.1	0.8	1.1	0.7
4	1.1	0.5	0.4	0.5	0	0.3	0.6	1.4	1.1	1.2	1.0	1.4	0.8
5	1.2	0.6	0.6	0.6	0.3	0	0.7	1.4	1.3	1.5	1.3	1.5	1.1
6	0.5	0.7	0.5	0.5	0.6	0.7	0	0.9	0.6	0.8	0.6	0.9	1.0
7	1.0	1.6	1.2	1.2	1.4	1.4	0.9	0	0.7	0.8	0.9	1.0	1.4
8	0.5	1.4	1.0	1.0	1.1	1.3	0.6	0.7	0	0.3	0.4	0.6	1.2
9	0.6	1.5	1.2	1.1	1.2	1.5	0.8	0.8	0.3	0	0.5	0.6	1.2
10	0.3	1.2	0.8	0.8	1.0	1.3	0.6	0.9	0.4	0.5	0	0.3	1.0
11	0.6	1.4	1.0	1.1	1.4	1.5	0.9	1.0	0.6	0.6	0.3	0	0.9
12	1.1	0.6	0.6	0.7	0.8	1.1	1.0	1.4	1.2	1.2	1.0	0.9	0
Polar	0	0	6	20	25	40	55	144	186	202	270	270	336
Dmd.	0	2	1	1	2	2	1	2	1	2	1	2	2
Early	0	0.5	0	1	0	3	0	1	2	1	0	3	3
Late	8	4	3	4	4	6	3	4	6	3	3	6	6
Serv.	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Table A.1: Routing Information for K12

Appendix B

Algorithms

B.1 The Additive Algorithm for Integer Programming

Prerequisite:

1. Convert all coefficients in the object function to positive.
2. Convert all constraints to $\leq$type, except those equations.
3. Assign slack variables for all the $\leq$constraints and equation constraints, where $S_{in} \geq 0$ and $S_{eqn} = 0$.

Step 1. (Initialization)

For i = 0 to N; j = 0 to N, k = 0 to K-1, let $x_{ijk} = 0$.
Let Z = some big value;
Candidates[ijk] = 1, for all i,j,k.

Step 2. (Slack infeasibility)

For all i, j, k combination available in the tree, calculate

$$I_{ijk} = \sum (negative\, slack\, value\, given\, x_{ijk} = 1)$$

Step 3. (Branching to one)

If $I_{ijk} = Max(I_{ijk})$ for all available i,j,k,d, then
let $x_{ijk} = 1$; candidates[ijk] = 0;

Go to Step 4.
Let $x_{ijk} = 0$;
Go to Step 3;
candidates[ijkd] = 1;
Else if all candidates[ijk] = 0, then
No more available branching variable, fathomed;
Return;
Go to Step 5.

Step 4 (Bounding and fathoming)

If z > Z, then
cannot yield a better solution, return;
Else Evaluate all slacks.
If infeasible then
go to Step 3. else calculate t_i. If

$$e_i \leq t_i \leq l_i$$

then it's a feasible solution. Record X and let Z = z.
Fathomed. Return.

Step 5 (Output result)

B&B complete. Output record and Z.
Go to Step 6.

Step 6. (End)

End.

B.2 The Push-Forward Insertion Heuristic (PFIH)

PFIH-1: Begin with an empty route starting from the depot. Set r = 1.

PFIH-2: If {all customers have been routed} then go to step 8;
else for all unrouted customer j: Compute the cost of inserting first node as in Eq. (5.1), and sort them in ascending order of their costs.

PFIH-3: Select the first customer j*, from the ordered list with the least cost and feasible in terms of time and capacity constraints.

PFIH-4: Append j* to the current route r and update the load of the route.

PFIH-5: For all unrouted customers j: For all edge {k, l} in the current route, compute the cost of inserting each of the unrouted customers between k and l.

PFIH-6: Select and unrouted customer j* and edge {k*, l*} that has the least cost.
If {insertion of customer j* between k* and l* is feasible in terms of time and capacity constraints}
then insert customer j* between k* and l*, update the load of the current route r and go to step 5.
else go to step 7.

PFIH-7: Begin a new route from the depot.
Set r = r + 1.
Go to step 2.

PFIH-8: All customers have been routed. Output results.
Stop PFIH.

B.3 The λ-interchange Local Search Descent Method

λ-int-1: Obtain a feasible solution S for the VRPTW using the PFIH.

λ-int-2: Select a solution $S \in N_{\lambda}(S)$ in the order indicated by 5.4.

λ-int-3: If {$C(S) < C(S)$}, then
accept S and go to Step 2,
else go to Step 4.

λ-int-4: If {neighborhood of $N_{\lambda}(S)$ has been completely searched (there are no moves that will result in a lower cost} then go to Step 5
else go to Step 2.

λ-int-5: Stop with the LSD solution.

B.4 The 2-interchange Simulated Annealing

T_s = Starting temperature of the SA method = 100
T_f = Final temperature of the SA method = 0.001
T_b = Temperature at which the current best solution was found
T_r = Reset temperature of the SA method, originally equal to T_s
T_k = Temperature of the current solution
S = Current solution
S_b = Current best solution
R = Number of resets to be done
τ = The time constant in the range of (0, 1).

SA-INT-1: Obtain a feasible solutions for the VRPTW using the PFIH.

SA-INT-2: Improve S using the 2-interchange LSD with Global-Best strategy.

SA-INT-3: Set cooling parameters: $T_s = T_b = T_r = T_k = 100$, $\tau = 0.5$.

SA-INT-4: Generate systematically an $S' \in N_2(S)$ by (2, 0) and (1, 0) operations, and compute $\Delta = C(S') - C(S)$.

SA-INT-5: If $\{(\Delta \leq 0)$ or $(\Delta > 0\,\text{and}\, \exp(-\Delta/T_k) \geq \theta)$, where θ is a random number between [0, 1]} then
set S = S'.
if {C(S) < C(S_b)} then
improve S using 2-interchange LSD (GB).
update $S_b = S$ and $T_b = T_k$.

SA-INT-6: Set k = k + 1. Update the temperature using Eq. (6.4).
If $\{N_2(S)$ is searched without any accepted move or $T_k = T_f\}$ then
reset $T_r = \max(T_r/2,\, T_b)$, and set $T_k = T_r$.

SA-INT-7: If {R resets have been made since the last S_b was found} then
go to Step SA-8.
else go to Step SA-4.

SA-INT-8: Terminate SA and print results.

B.5 The Strict Tabu (S-TABU) Algorithm

S-TABU1: Obtain an initial solution by PFIH and update the global best solution with the present solution;

S-TABU2: Initialize tabu move list and tabu solution list as well as candidate list, add current solution to the tabu solution list;

S-TABU3: Do intensification, a 2-interchange (GB) procedure;

S-TABU4: Do diversification, and record random solutions encountered during random hops in the candidate list;

S-TABU5: If not diversified and total number of iterations is less than MAX_ITERATION, go to S-TABU3;

S-TABU6: Else improve the global best solution obtained so far by 2-interchange (GB) and return the improved solutions;

S-TABU7: Terminate S-TABU and print result.

B.6 A Genetic Algorithm Implementation

GA-1: Generate initial population of N chromosomes (partly from PFIH and its mutation and partly from totally random selection). Calculate time precedence according the latest arrival time of each node;

GA-2: Evaluate the fitness f(x) of each chromosome x in the population, calculate average fitness and standard deviation, thus set mutation probability;

GA-3: Create a new population by repeating following steps until the new population is complete;
1.[*Selection*] Select two parent chromosomes from a population by Roulette wheel or tournament selection;
2.[*Crossover*] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents;
3.[*Mutation*] With a mutation probability mutate new offspring at a random locus (position in chromosome);

4.[*Accepting*] Place new offspring in a new population;
5.[*Hill Climbing*] Performing hill climbing with 1-neighborhood first-best search;
6.[*Recovery*] Replace the worse 4% chromosomes in the new population with the best 4% in the parents population;

GA-4: Update the old population with the newly generated population;

GA-5: If the certain number of generation is reached, stop, perform a 2-interchange (GB) on the best solution in the current population and return the improved solution;

GA-6: Else go to step 2

Appendix C

Tables of Comparison

In this chapter, experimental results from all the heuristics we have attempted against the best published results[1] are tabulated and presented. For deterministic methods, only the best solutions are included and for stochastic methods, both the best and the average solutions are included.[2] All best solutions obtained by the author are shown with the number of vehicles used and the total cost. Average results taken from 8 trials each, however, are shown only with the average total costs rounded off to integer values. Our solutions which are equivalent or better than the best published results are highlighted. The average times indicated here are in terms of Pentium-II 266 MMX seconds, and are for reference only.

Table C.1: The Comparison of Best Results

Prob.	**Best Pub.**	**2-INT**	**SA**	**TABU**	**GA**
C101	10 827	**10 828.937**	**10 828.937**	**10 828.937**	**10 828.937**
C102	10 827	10 923.375	10 923.375	10 901.525	10 868.798
C103	10 828.06	10 994.87	10 994.87	10 895.774	11 939.456
C104	10 824.78	11 1130.85	11 1130.85	10 887.493	10 959.346
C105	10 829	**10 828.937**	**10 828.937**	**10 828.937**	**10 828.937**
C106	10 827	10 1052.07	10 1052.07	10 941.154	**10 828.937**

[1]Many best published results in C problem set have round-off errors, therefore appear slightly better than our results but actually they are the equivalent.

[2]Although most of the solutions here follow the parameter settings in the main chapters, some of them, however were obtained by using simplified model or more iterations. Most solutions are probablistic, therefore might not be replicated.

C107	10 829	10 875.62	10 867.234	**10 828.937**	**10 828.937**
C108	10 827	10 878.089	10 876.427	**10 828.937**	**10 828.937**
C109	10 829	10 1100.71	10 1124.44	**10 828.937**	**10 828.937**
C201	3 590	**3 591.557**	**3 591.557**	**3 591.557**	**3 591.557**
C202	3 590	4 801.281	4 787.856	4 704.51	3 631.146
C203	3 591.55	4 1225.1	4 1208.94	4 727.221	4 745.934
C204	3 590.6	3 661.213	3 642.691	**3 590.599**	3 604.998
C205	3 589	3 625.333	3 613.327	**3 588.876**	**3 588.876**
C206	3 588	3 704.162	3 694.25	**3 588.493**	**3 588.493**
C207	3 588	3 725.404	3 704.365	3 600.841	3 593.195
C208	3 588	3 902.597	3 888.685	**3 588.324**	3 590.873
R101	18 1608	20 1847.37	20 1847.37	20 1707.95	20 1676.86
R102	17 1434	19 1720.46	18 1544.82	16 1488.59	18 1558.59
R103	13 1207	17 1551.34	17 1482.39	15 1293.85	15 1311.81
R104	10 982.01	12 1184.38	12 1184.38	11 1057.02	12 1128.29
R105	14 1377.11	17 1671.94	17 1595.68	16 1431.56	17 1496.37
R106	12 1252.03	14 1439	14 1434.3	14 1310.02	14 1352.9
R107	11 1126.69	13 1390.55	12 1270.04	12 1174.89	13 1240.82
R108	10 968.59	12 1256.61	12 1186.34	11 1024.93	11 1074.67
R109	11 1205	14 1525.45	14 1515.38	14 1250.5	15 1300.29
R110	11 1080.36	13 1444.63	13 1430.35	13 1179	14 1295.77
R111	10 1104.83	13 1370.64	13 1370.21	13 1148	13 1185.02
R112	10 953.63	12 1215.27	12 1180.18	11 1061.4	12 1096.21
R201	4 1354	5 1791.42	5 1726.13	5 1385.69	**8 1329.74**
R202	3 1530.49	4 1610.02	4 1581.64	**4 1233.37**	**7 1307.03**
R203	3 1126	4 1475.63	4 1248.55	**4 1081.04**	**6 1086.43**
R204	2 914	3 1098.67	3 1088.06	**3 895.867**	6 956.384
R205	3 1128	4 1370.68	4 1344.2	**4 1121.03**	5 1131.18
R206	3 833	3 1341.22	3 1369.5	3 1089.48	5 1187.25
R207	3 904	3 1167.11	3 1153.65	3 949.101	4 987.703
R208	2 759.21	3 988.367	3 971.572	3 793.233	3 845.937
R209	2 855	4 1210.96	4 1206.58	4 1110.3	5 1097.42

R210	3 1052	4 1312.91	4 1238.14	4 1071.3	6 1098.69
R211	3 816	3 1232.48	3 1140.65	3 914.723	7 932.483
RC101	14 1669	17 1948.94	17 1940.57	16 1734.18	17 1728.37
RC102	12 1554.75	16 1803.95	16 1777.02	14 1562.62	17 1603.53
RC103	11 1110	14 1627.02	14 1620.35	13 1377.93	14 1501
RC104	10 1135.83	12 1408.5	12 1408.5	12 1230.95	12 1276.02
RC105	14 1602	17 1920.19	17 1809.78	**16 1597.67**	17 1688.77
RC106	11 1448.26	14 1656.29	14 1645.24	14 1455.73	14 1491.58
RC107	11 1230.54	15 1677.92	15 1653.65	13 1371.3	14 1434.22
RC108	10 1139.82	12 1377.93	13 1335.06	12 1218.61	12 1333.15
RC201	4 1249	5 2070.4	5 1891.9	5 1617.5	10 1565.67
RC202	4 1221	5 1970.66	5 1956.97	5 1429.04	10 1353.27
RC203	3 1203	4 1627.74	4 1522.68	**4 1179.67**	**6 1189.06**
RC204	3 879	4 1314.21	4 1290.89	4 939.678	4 989.943
RC205	4 1389	5 1923.9	5 1907	5 1487.49	9 1465.8
RC206	3 1213	4 1663.87	4 1645.17	4 1357.32	5 1388.13
RC207	3 1181	4 1539.85	4 1497.54	4 1295.9	6 1304.48
RC208	3 919	3 1490.27	3 1422.94	3 1040.47	6 1003.43

Table C.2: 2-Interchange Results

Prob.	**Best Pub.**	**2-INT(FB)**	**Time**	**2-INT(GB)**	**Time**
C101	10 827	**10 828.937**	32	**10 828.937**	7
C102	10 827	10 923.484	33	10 923.375	25
C103	10 828.06	10 1156.27	33	10 994.87	70
C104	10 824.78	11 1156.14	29	11 1130.85	69
C105	10 829	**10 828.937**	31	**10 828.937**	6
C106	10 827	10 1053.54	27	10 1052.07	8
C107	10 829	10 875.62	31	10 875.62	12
C108	10 827	10 921.974	30	10 878.089	23
C109	10 829	10 1100.71	29	10 1172.33	9
C201	3 590	**3 591.557**	39	**3 591.557**	12

C202	3 590	4 866.523	38	4 801.281	33
C203	3 591.55	4 1232.72	36	4 1225.1	103
C204	3 590.6	4 690.149	43	3 661.213	42
C205	3 589	3 680.276	44	3 625.333	62
C206	3 588	4 717.958	40	3 704.162	123
C207	3 588	4 749.786	48	3 725.404	46
C208	3 588	4 956.63	42	3 902.597	70
R101	18 1608	21 2007.18	39	20 1847.37	35
R102	17 1434	19 1773.24	38	19 1720.46	46
R103	13 1207	17 1600.47	32	17 1551.34	55
R104	10 982.01	12 1203.36	28	12 1184.38	65
R105	14 1377.11	16 1763.76	34	17 1671.94	37
R106	12 1252.03	14 1519.27	27	14 1439	45
R107	11 1126.69	13 1420.56	27	13 1390.55	46
R108	10 968.59	12 1328.46	29	12 1256.61	52
R109	11 1205	14 1590.57	26	14 1525.45	39
R110	11 1080.36	13 1513.34	28	13 1444.63	45
R111	10 1104.83	13 1415.62	27	13 1370.64	34
R112	10 953.63	12 1215.27	38	12 1225.54	56
R201	4 1354	5 1932.12	39	5 1791.42	76
R202	3 1530.49	4 1667.47	40	4 1610.02	48
R203	3 1126	4 1562.03	42	4 1475.63	100
R204	2 914	3 1129.82	42	3 1098.67	149
R205	3 1128	4 1470.02	38	4 1370.68	47
R206	3 833	3 1341.22	42	3 1377.43	52
R207	3 904	3 1209.64	51	3 1167.11	142
R208	2 759.21	3 1011.13	51	3 988.367	185
R209	2 855	4 1392	42	4 1210.96	128
R210	3 1052	4 1474.9	43	4 1312.91	121
R211	3 816	3 1234.45	46	3 1232.48	28
RC101	14 1669	18 2130.76	42	17 1948.94	38
RC102	12 1554.75	17 2034.43	35	16 1803.95	83

RC103	11 1110	14 1664.41	34	14 1627.02	49
RC104	10 1135.83	12 1455.97	28	12 1408.5	53
RC105	14 1602	17 2049.34	30	17 1920.19	51
RC106	11 1448.26	14 1744.72	27	14 1656.29	29
RC107	11 1230.54	15 1737.62	27	15 1677.92	42
RC108	10 1139.82	12 1377.93	28	13 1394.28	47
RC201	4 1249	5 2129.31	39	5 2070.4	56
RC202	4 1221	5 2000.65	29	5 1970.66	16
RC203	3 1203	4 1648.87	29	4 1627.74	100
RC204	3 879	4 1314.21	42	4 1314.21	18
RC205	4 1389	5 2133.07	34	5 1923.9	109
RC206	3 1213	4 1841.27	35	4 1663.87	60
RC207	3 1181	4 1665.99	50	4 1539.85	106
RC208	3 919	3 1490.27	40	3 1490.27	16

Table C.3: Simulated Annealing Results

Prob.	**Best Pub.**	**Saint**	**Avg.**	**Time**	**Sarand**	**Avg.**	**Time**
C101	10 827	**10 828.937**	829	8	**10 828.937**	829	10
C102	10 827	10 923.375	924	100	10 923.375	926	24
C103	10 828.06	10 955.079	955	56	10 994.87	998	63
C104	10 824.78	11 1118.1	1121	120	11 1130.85	1135	65
C105	10 829	**10 828.937**	829	60	**10 828.937**	829	10
C106	10 827	10 933.577	935	136	10 1052.07	1053	12
C107	10 829	10 867.234	868	117	10 875.62	876	16
C108	10 827	10 868.089	868	91	10 878.089	880	27
C109	10 829	10 1157.54	1158	240	10 1172.33	1224	66
C201	3 590	**3 591.557**	592	382	**3 591.557**	592	17
C202	3 590	4 787.856	788	111	4 801.281	803	37
C203	3 591.55	4 1137.82	1139	171	4 1225.1	1225	65
C204	3 590.6	3 642.691	650	85	3 661.213	661	54
C205	3 589	3 613.327	621	112	3 625.333	627	71
C206	3 588	3 694.25	694	172	3 704.162	707	139

C207	3 588	3 704.365	704	172	3 725.404	728	55
C208	3 588	3 888.685	889	126	3 902.597	906	72
R101	18 1608	20 1847.37	1847	33	20 1847.37	1849	34
R102	17 1434	18 1544.82	1548	96	18 1720.46	1724	42
R103	13 1207	17 1482.39	1490	78	17 1551.34	1553	48
R104	10 982.01	12 1184.38	1184	65	12 1184.38	1186	59
R105	14 1377.11	17 1595.68	1596	84	17 1671.94	1674	41
R106	12 1252.03	14 1434.3	1434	64	14 1439	1440	48
R107	11 1126.69	12 1270.04	1279	102	13 1390.55	1391	52
R108	10 968.59	12 1186.34	1188	107	12 1256.61	1259	58
R109	11 1205	14 1515.38	1518	61	14 1525.45	1531	45
R110	11 1080.36	13 1430.35	1430	80	13 1444.63	1445	51
R111	10 1104.83	13 1370.21	1370	60	13 1370.64	1372	40
R112	10 953.63	12 1180.18	1180	110	12 1225.54	1226	59
R201	4 1354	5 1726.13	1726	169	5 1786.54	1789	86
R202	3 1530.49	4 1581.64	1581	146	4 1610.02	1610	57
R203	3 1126	4 1248.55	1248	406	4 1475.63	1476	113
R204	2 914	3 1088.06	1088	271	3 1098.67	1099	116
R205	3 1128	4 1344.2	1344	157	4 1370.68	1374	54
R206	3 833	3 1369.5	1370	98	3 1377.43	1379	60
R207	3 904	3 1153.65	1154	218	3 1167.11	1168	156
R208	2 759.21	3 971.572	972	271	3 988.367	988	187
R209	2 855	4 1206.58	1207	186	4 1210.96	1212	140
R210	3 1052	4 1238.14	1241	272	4 1312.91	1313	128
R211	3 816	3 1140.65	1141	185	3 1232.48	1233	32
RC101	14 1669	17 1940.57	1941	57	17 1948.94	1951	38
RC102	12 1554.75	16 1777.02	1779	68	16 1803.95	1806	45
RC103	11 1110	14 1620.35	1620	67	14 1627.02	1634	50
RC104	10 1135.83	12 1408.5	1409	60	12 1408.5	1409	58
RC105	14 1602	17 1809.78	1809	79	17 1920.19	1922	41
RC106	11 1448.26	14 1645.24	1645	37	14 1656.29	1659	23
RC107	11 1230.54	15 1653.65	1654	76	15 1677.92	1679	47

RC108	10 1139.82	13 1394.28	1398	56	13 1335.06	1376	53
RC201	4 1249	5 1891.9	1892	173	5 2070.4	2071	58
RC202	4 1221	5 1956.97	1957	57	5 1970.66	1971	21
RC203	3 1203	4 1522.68	1523	220	4 1627.74	1629	112
RC204	3 879	4 1290.89	1291	83	3 1314.21	1314	27
RC205	4 1389	5 1907	1908	192	5 1923.9	1929	121
RC206	3 1213	4 1645.17	1645	164	4 1663.87	1666	69
RC207	3 1181	4 1497.54	1498	168	4 1539.85	1542	94
RC208	3 919	3 1422.94	1423	111	3 1490.27	1493	20

Table C.4: Tabu Search Results

Prob.	**Best Pub.**	**S-Tabu**	**Avg.**	**Time**	**Int-Tabu**	**Avg.**	**Time**
C101	10 827	**10 828.937**	829	1211	**10 828.937**	834	455
C102	10 827	10 901.527	906	1202	10 912.651	913	319
C103	10 828.06	10 895.774	897	1402	10 998.152	998	412
C104	10 824.78	10 887.493	899	1117	11 1062.65	1104	536
C105	10 829	**10 828.937**	829	1210	**10 828.937**	829	435
C106	10 827	10 941.154	941	1543	10 1025.81	1025	450
C107	10 829	**10 828.937**	829	1504	**10 828.937**	829	452
C108	10 827	**10 828.937**	829	1400	**10 828.937**	829	534
C109	10 829	**10 828.937**	829	259	10 857.359	860	230
C201	3 590	**3 591.557**	592	59	**3 591.557**	592	595
C202	3 590	4 704.51	737	2612	4 812.047	812	620
C203	3 591.55	4 727.221	733	2812	4 922.14	933	526
C204	3 590.6	**3 590.599**	601	1534	4 653.66	720	640
C205	3 589	**3 588.876**	589	860	3 622.712	623	608
C206	3 588	**3 588.493**	592	810	4 677.087	677	420
C207	3 588	3 600.841	624	3192	4 736.417	736	542
C208	3 588	**3 588.324**	622	1807	4 691.488	941	440
R101	18 1608	20 1707.95	1734	2551	20 1791.09	1835	1117
R102	17 1434	16 1488.59	1538	640	19 1719.37	1757	616

R103	13 1207	15 1293.85	1309	811	15 1309.93	1412	665
R104	10 982.01	11 1057.02	1066	957	11 1120.57	1130	610
R105	14 1377.11	16 1431.56	1450	809	17 1629.07	1635	587
R106	12 1252.03	13 1310.02	1370	609	14 1450.86	1456	456
R107	11 1126.69	12 1174.89	1251	1035	13 1329.99	1380	412
R108	10 968.59	11 1024.93	1054	1133	11 1135.66	1158	467
R109	11 1205	14 1250.5	1276	1047	14 1256.36	1385	422
R110	11 1080.36	13 1179	1181	974	14 1332.2	1375	433
R111	10 1104.83	13 1148	1208	1129	13 1225.76	1284	560
R112	10 953.63	12 1061.4	1063	1211	11 1075.95	1088	600
R201	4 1354	5 1385.69	1410	2777	5 1472.8	1488	1872
R202	3 1530.49	**4 1233.37**	1251	3601	**5 1296.22**	1324	1728
R203	3 1126	**4 1081.04**	1115	3925	4 1235.16	1242	1787
R204	2 914	**3 895.867**	924	3151	3 971.016	1032	1754
R205	3 1128	**4 1121.03**	1148	3162	4 1230.46	1244	1632
R206	3 833	3 1089.48	1125	3486	4 1174.53	1229	1555
R207	3 904	3 949.101	1047	4025	3 1069.04	1112	1964
R208	2 759.21	3 793.233	820	3591	3 925.309	932	1879
R209	2 855	4 1110.3	1166	2778	4 1231.34	1244	1710
R210	3 1052	4 1071.3	1123	2961	4 1142.25	1163	1846
R211	3 816	3 914.723	942	3097	3 1048.1	1067	1929
RC101	14 1669	16 1734.18	1752	1241	16 1814.31	1888	431
RC102	12 1554.75	14 1562.62	1596	1210	14 1674.9	1792	487
RC103	11 1110	13 1377.93	1416	993	13 1421.49	1473	330
RC104	10 1135.83	12 1230.95	1260	1045	11 1284.13	1308	400
RC105	14 1602	**16 1597.67**	1603	808	17 1698	1712	595
RC106	11 1448.26	14 1455.73	1488	743	14 1529.69	1532	450
RC107	11 1230.54	13 1371.3	1399	1000	15 1650.12	1650	610
RC108	10 1139.82	12 1218.61	1250	1089	12 1279.26	1320	599
RC201	4 1249	5 1617.53	1638	2574	5 1727.6	1728	1946
RC202	4 1221	5 1429.04	1438	2799	5 1437.51	1455	1533
RC203	3 1203	**4 1179.67**	1207	3314	4 1244.3	1245	1438

RC204	3 879	4 939.678	1109	4037	4 1071.07	1187	1551
RC205	4 1389	5 1487.49	1512	2505	6 1505.65	1514	1430
RC206	3 1213	4 1357.32	1367	3271	4 1390.88	1411	1544
RC207	3 1181	4 1295.9	1321	3282	4 1317.86	1319	1409
RC208	3 919	3 1040.47	1052	3957	3 1256.98	1315	1633

Table C.5: Genetic Algorithm Results

Problems	**Best Pub.**	**GA-PMX**	**Avg.**	**Time**	**GA-H1M2**	**Avg.**	**Time**
C101	10 827	**10 828.937**	829	525	**10 828.937**	829	522
C102	10 827	10 868.798	883	616	10 895.226	904	587
C103	10 828.06	11 939.456	974	645	11 957.268	980	539
C104	10 824.78	10 984.064	1015	564	10 959.346	992	528
C105	10 829	**10 828.937**	856	512	**10 828.937**	830	535
C106	10 827	10 829.397	867	497	**10 828.937**	829	496
C107	10 829	**10 828.937**	829	508	**10 828.937**	829	504
C108	10 827	**10 828.937**	829	511	**10 828.937**	829	749
C109	10 829	**10 828.937**	852	519	**10 828.937**	834	543
C201	3 590	**3 591.557**	592	1053	**3 591.557**	592	1249
C202	3 590	4 725.638	743	800	3 631.146	674	874
C203	3 591.55	4 745.934	850	842	4 758.625	832	749
C204	3 590.6	3 604.998	606	1057	3 619.524	620	988
C205	3 589	**3 588.876**	589	1072	**3 588.876**	589	954
C206	3 588	**3 588.493**	594	949	3 609.032	621	1012
C207	3 588	3 652.836	653	1046	3 593.195	594	1828
C208	3 588	4 637.572	658	951	3 590.873	606	932
R101	18 1608	20 1676.86	1699	445	21 1757.09	1786	518
R102	17 1434	18 1558.59	1569	453	19 1603.83	1665	455
R103	13 1207	15 1311.81	1334	702	16 1432.29	1461	476
R104	10 892.01	12 1128.29	1162	497	12 1166.57	1181	565
R105	14 1377.11	17 1496.37	1510	666	17 1552.85	1591	451
R106	12 1252.03	14 1352.9	1369	701	15 1419.3	1424	481
R107	11 1126.69	13 1240.82	1263	489	14 1295.25	1331	455

R108	10 968.59	12 1091.69	1121	360	11 1074.67	1158	666
R109	11 1205	15 1300.29	1328	405	15 1322.82	1410	433
R110	11 1080.36	14 1295.77	1323	452	14 1332.16	1345	463
R111	10 1104.83	13 1185.02	1211	460	13 1245.7	1263	474
R112	10 953.63	12 1096.21	1104	464	12 1125.59	1140	502
R201	4 1354	**8 1329.74**	1402	654	10 1463.29	1476	609
R202	3 1530.49	**7 1307.03**	1335	601	**7 1410.93**	1454	802
R203	3 1126	**6 1086.43**	1134	785	8 1145.19	1172	701
R204	2 914	6 956.384	978	959	5 983.81	1010	899
R205	3 1128	5 1131.18	1154	913	6 1238.72	1289	760
R206	3 833	6 1187.25	1202	884	5 1254.19	1268	739
R207	3 904	4 987.703	1021	1100	7 1071.3	1112	742
R208	2 759.21	3 846.777	880	796	3 824.542	848	986
R209	2 855	5 1111.8	1132	895	5 1097.42	1188	808
R210	3 1052	6 1098.69	1132	901	7 1157.67	1171	714
R211	3 816	5 971.804	990	879	7 932.483	1002	719
RC101	14 1669	17 1728.37	1770	438	17 1791.35	1857	695
RC102	12 1554.75	17 1603.53	1664	456	16 1662.17	1706	355
RC103	11 1110	14 1502.76	1532	505	14 1501	1576	488
RC104	10 1135.83	12 1276.02	1288	465	12 1329.64	1401	505
RC105	14 1602	17 1688.77	1755	446	18 1860.6	1872	443
RC106	11 1448.26	14 1491.58	1502	447	14 1560.02	1567	439
RC107	11 1230.54	14 1434.22	1496	446	15 1547.25	1605	463
RC108	10 1139.82	13 1344.27	1365	253	12 1333.15	1366	494
RC201	4 1249	10 1565.67	1598	601	12 1660.24	1696	555
RC202	4 1221	10 1353.27	1421	608	10 1479.52	1502	331
RC203	3 1203	**6 1189.06**	1239	1202	7 1215.19	1346	676
RC204	3 879	4 989.943	1054	982	5 1026.47	1045	755
RC205	4 1389	9 1465.8	1512	643	11 1649.53	1660	556
RC206	3 1213	6 1428.72	1429	768	5 1388.13	1402	750
RC207	3 1181	5 1308.61	1321	862	6 1304.48	1448	743
RC208	3 919	4 1143.89	1172	1015	6 1003.43	1038	712

Bibliography

[1] Bachem, A., B. Stechemetz and M. Wottawa, *An Efficient Parallel Cluster-heuristic for Large Traveling Salesman Problems.* Working paper (1994).

[2] Battiti, R. and G. Tecchiolli, *The Reactive Tabu Search.* ORSA Journal on Computing (1994).

[3] Bersimas, D. and D. Simchi-Levi, *A New Generation of Vehicle Routing Research, Robust Algorithms, Addressing Uncertainty.* Working paper (1993).

[4] Blanton, J. L. Jr. and R. L. Wainwright, *Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms.* Proceedings of the Fifth International Conference on Genetic Algorithms (1993).

[5] Bodin, L., B. Golden, A. Assad and M. Ball, *Routing and Scheduling of Vehicles and Crews - the State of the Art.* Pergamon Press (1983).

[6] Borches, B., *Using an Interior Point Method in a Branch and Bound Algorithm for Integer Programming*. Working paper (1992).

[7] Braca, J., J. Bramel, B. Posner and D. Simchi-Levi, *A Computerized Approach to New York City School Bus Routing Problem.* Working paper (1994).

[8] Bramel, J. and D. Simchi-Levi, *Probabilistic Analyses and Practical Algorithms for Vehicle Routing Problems with Time Windows.* Working paper (1993).

[9] Clarkson, Kenneth L., *Las Vegas Algorithms for Linear and Integer Programming When the Dimension is Small.* Working paper (1995)

[10] Desrochers, M., J. Desrosier and M. Solomon, *A New Optimization Algorithm for Vehicle Routing Problems with Time Windows.* Operations Research 40 (1992).

[11] Desrosier, J., Y. Dumas, M. Solomon and F. Soumis, *Time Constraint Routing and Scheduling.* Handbooks on Operations Research and management Science, Volume Network, North Holland, Amsterdam (1995).

[12] Eilon, S., C. Watson-Grandy and N. Christofides, *Distribution Management: Mathematical Modeling and practical Analysis.* Hafner, New York (1971).

[13] Fisher, M. L., K. O. Jornsten and O. B. G. Madsen, *Vehicle Routing with Time Windows.* Working paper (1992).

[14] Fisher, M. L. , *Vehicle Routing.* Handbooks in Operations Research & Management Science, Vol 8 (1995).

[15] Glover, F., *Tabu Search - Part I.* ORSA Journal on Computing (1989).

[16] Glover, F., Manuel Laguna, *Tabu Search.* Kluwer Academic Publishers (1997).

[17] Golden, B. and A. Assad, *Vehicle Routings: Methods and Studies.* North Holland, Amsterdam (1988).

[18] Hamacher, A. and C. Moll, *A New Heuristic for Vehicle Routing with Narrow Time Windows.* Working paper (1995).

[19] Hamacher, A., W. Hochstattler and C. Moll, *Tree Partitioning under Constraints Clustering for Vehicle Routing Problems.* Working paper (1997).

[20] Holland, J. H., *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor(1975).

[21] Homaifar, A., S. Guan and G. E. Liepins, *A New Approach to the Traveling Salesman Problem by Genetic Algorithms.* Proceedings of the Fifth International Conference on Genetic Algorithms (1993).

[22] Kelly, J. P. and J. Xu, *A Generic Set-Covering-Based Heuristic for Vehicle Routing Problems.* Working paper (1997).

[23] Kirkpatrick, S., C.D. Gelatt Jr. and M.P. Vecchi, *Optimization by Simulated Annealing.* Science, 20:671-680 (1983)

[24] Kohl, N. and O. Madson, *An Optimization Algorithm for the Vehicle Routing Problem with Time Windows based on Lagrangean Relaxation.* Working paper (1995).

[25] Kolen, A. W. J., A. H. G. R. Kan and H. W. J. M. Trienekens, *Vehicle Routing with Time Windows.* Operations Research 35 (1987).

[26] Linderoth, J. T. and M. W. P. Savelsbergh, *A Computational Study of Search Strategies for Mixed Integer Programming.* Working paper (1997).

[27] Metropolis, N., A.W. Rosenbluth, M.N. Rosenbluth, A.H. *Teller and E. Teller, Equations of state calculations by fast computing machines.* The Journals of Chemical Physics, 21:1087-1092 (1953).

[28] Oliver, I. M. , D. J. Smith and J. R. C. Holland, *A Study of Permutation Crossover Operations on the Traveling Salesman Problem.* Proceedings of the Fourth International Conference on Genetic Algorithms (1991).

[29] Osman, Ibrahim H. and N. Christofides, *Simulated Annealing and Descent Algorithms for Capacitated Clustering Problem.* Research Report, Imperial College, University of London (1989).

[30] Osman, I. H., *Meta-strategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem.* Annuals Operations Research 41 (1993).

[31] Pincus, M., *A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems.* Operations Research, 18:1225-1228 (1970).

[32] Prinetto, P., M. Rebaudengo and M. Sonza Reorda, *Hybrid Genetic Algorithms for the Traveling Salesman Problem.* Proceedings of the Fifth International Conference on Genetic Algorithms (1993).

[33] Savelsbergh, M. W. P., *Local Search for Routing Problems with Time Windows.* Annuals of Operating Research (1985).

[34] Schildt, Herbert, *C++: The Complete Reference.* McGraw-Hill (1991).

[35] Shaw, Paul, *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems.* Working paper (1998).

[36] Sieksma, Gerard, *Linear and Integer Programming, Theory and Practice.* Marcel Dekker Inc.

[37] Solomon, M. M., *Algorithms for Vehicle Routing and Scheduling Problems with Time Window Constraints.* Operations Research 35 (2) (1987).

[38] Stroustrup, B., *C++ Programming Language.* Addison-Wesley Publishing Co., 2nd ed. (1997).

[39] Taha, H. A., *Operation Research, an Introduction.* Prentice Hall International, 6th ed.

[40] Thangiah, S. R., I. H. Osman and T. Sun, *Algorithms for the Vehicle Routing Problems with Time Deadlines.* Working paper (1992).

[41] Thangiah, S. R., I. H. Osman and T. Sun, *Hybrid Genetic Algorithm, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows.* Working paper (1993).

[42] Thangiah, S. R. , *An Adaptive Clustering Method using a Geometric Shape for Vehicle Routing Problems with Time Windows.* Proceedings of the Sixth International Conference on Genetic Algorithms (1995)

[43] Thangiah, S. R., *Vehicle Routing with Time Windows using Genetic Algorithms*, working paper (1995).

[44] Whitney, D., T. Starkweather and D. Fuquay, *Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator.* Proceedings of the Fourth International Conference on Genetic Algorithms (1991).

[45] Xu, J. and J. P. Kelly, *A Network Flow-based Tabu Search Heuristic for the Vehicle Routing Problem.* Working paper (1996).