UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

DESIGN AND IMPLEMENTATION OF AN INTEGRATED ALGORITHM FOR THE VEHICLE ROUTING PROBLEM WITH MULTIPLE CONSTRAINTS

ALWYN JAKOBUS MOOLMAN

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

MASTER OF SCIENCE (INDUSTRIAL SYSTEMS)

in the

FACULTY OF ENGINEERING, BUILT ENVIRONMENT AND
INFORMATION TECHNOLOGY

UNIVERSITY OF PRETORIA

APRIL 2004

ABSTRACT

Title: Design and Implementation of an Integrated Algorithm for the Vehicle Routing Problem with Multiple Constraints.

Author: Alwyn Jakobus Moolman

Promotor: Professor VSS Yadavalli

Department: Industrial and Systems Engineering

University: University of Pretoria

Degree: Master of Science (Industrial Systems)

Supply Chain Management could be defined as the practice of analysing all aspects of acquiring, storing, moving, and delivering materials from the time they are acquired through any conversion or production processes through to the time final products are used or sold. Due to the expansion of online retail and online Business To Business (B2B) transactions, there is a great need for companies to invest in effective solutions that will aid them in ensuring that their supply chains, and particularly the distribution side of the supply chain, work as effectively and seamlessly as possible. A company will battle to be successful if it has the best products but a poor fulfilment side to its business. It is evident that there are at present a number of shortfalls within the fulfilment environment. There is thus scope for an effective, all-encompassing order fulfilment engine that addresses all problems, and ensures that all the positive aspects are maintained.

There exist a need in the industry for an affordable service, which can assist with the optimization of distribution routes. Not all businesses have a large enough fleet to verify the costs associated with a fleet management system that includes optimization. Such a system normally requires a skilled operator, that add to the cost. A solution to this problem is the implementation of an optimization server

in public domain. This is done by implementing a routing engine on an Application Service Provider (ASP). The ASP is a web-enabled distribution and fulfilment planning and optimization system that assist its users in the fulfilment of their customers' orders. This allows the provider to manage the system from one centralised server that allows other users to access the system via the Internet.

The Vehicle Routing Problem (VRP) is an important problem occurring in many distribution systems. VRP can be described as the problem of designing least cost routes from one depot to a set of geographically scattered points. The basic VRP is not sufficient enough for implementing in distribution systems. Additional constraints such as multiple time windows, heterogeneous fleet, double scheduling, stop priority and route length must be added to the basic problem.

Designing an algorithm that is efficient to solve the VRP with the required additional constraints, as well as effective in an ASP environment involves the extension of existing methods as well as designing new ones. This research implements a Tabu Search heuristic in a two-stage process to solve the problem. The Tabu Search was selected because of its memory capabilities.

Key words: Fulfilment; Vehicle Routing Problem; Application Service Provider; Tabu Search; Multiple constraints; Multiple Time Windows; Heterogeneous fleet; Double scheduling; Supply Chain Management; Business to Business.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

AMP. Adaptive Memory Procedure.

ASP. Application Service Provider.

GA Genetic algorithms

MDVRP Multi-Depot Vehicle Routing Problem

NP-hard Non-polynomial hard

SA Simulate annealing

SIH Sequential Insertion Heuristic

TS Tabu search

TSP Travelling Salesmen Problem

VFM Vehicle Fleet Mix

VRP Vehicle Routing Problem

VRPM Vehicle Routing Problem with multiple uses of vehicles

VRPMC Vehicle Routing Problem with Multiple Constraints

VRPHE Vehicle Routing Problem with Heterogeneous Fleet

Chapter 1

1 INRODUCTION

1.1. Overview

Supply Chain Management could be defined as the practice of analysing all aspects of acquiring, storing, moving, and delivering materials from the time they are acquired through any conversion or production processes through to the time final products are used or sold. A company's supply chain may consists of geographically dispersed facilities where raw materials, intermediate products, or finished products are acquired, transformed, stored, or sold, and transportation links connecting the facilities along which products flow.

Supply Chain Management thus involves whatever an organisation does to plan, source, make and deliver its products.

There is a distinction between manufacturing facilities and distribution centres. In manufacturing facilities, physical product transformations take place and at distribution centres, products are received, sorted, put into inventory, then picked from inventory and dispatched. These products are not physically transformed.

The company's goal is to add value to its products as they pass through its supply chain and transport them to geographically dispersed markets in the correct quantities, with the correct specifications, at the correct time, and at a competitive cost.

Supply chain management crystallises those concepts of integrated business planning that have been espoused for many years by logistics experts, strategists,

and operations research practitioners. Today, integrated planning is possible due to advances in Information Technology (IT).

Due to the expansion of online retail and online Business To Business (B2B) transactions, there is a great need for companies to invest in effective solutions that will aid them in ensuring that their supply chains, and particularly the distribution side of the supply chain, work as effectively and seamlessly as possible. A company will battle to be successful if it has the best products but a poor fulfilment side to its business. Without effective fulfilment, customers will not be satisfied and hence all confidence in that particular company will be lost. Many online retailing ventures have failed solely due to the fact that their fulfilment systems were not effective enough and traditional brick and mortar companies have under-optimized fulfilment systems where great improvements are possible.

Current predictions are that business-to-business (B2B) online trading will grow from US$336 billion in 2001 to US$6.3 trillion in 2005.

There is an indication that online supply chains will dominate the B2B commerce arena, swelling from 3% currently to 42% of the total B2B USA trade over the next 5 years. Specifically, five major industries – aerospace and defence, chemicals, computer and telecommunications equipment, electronics, and motor vehicles and parts – will conduct more than half of the B2B transactions online by 2004. Computers and telecommunications will become the biggest online B2B market, with sales soaring past US$1 trillion by 2005. The other four areas will each top US$500 billion by 2005. [1]

[1] Source: Jupiter Research [24] (p. 13)

From the above statistics, this market is destined for phenomenal growth and hence there is great scope for any products or business ventures related to ensuring that this environment operates optimally.

The focus of this thesis is on fulfilment operations, specifically routing and distribution, but other factors that influence fulfilment must be taken into account.

The following diagram illustrate fulfilment within the supply chain and factors that influence it:

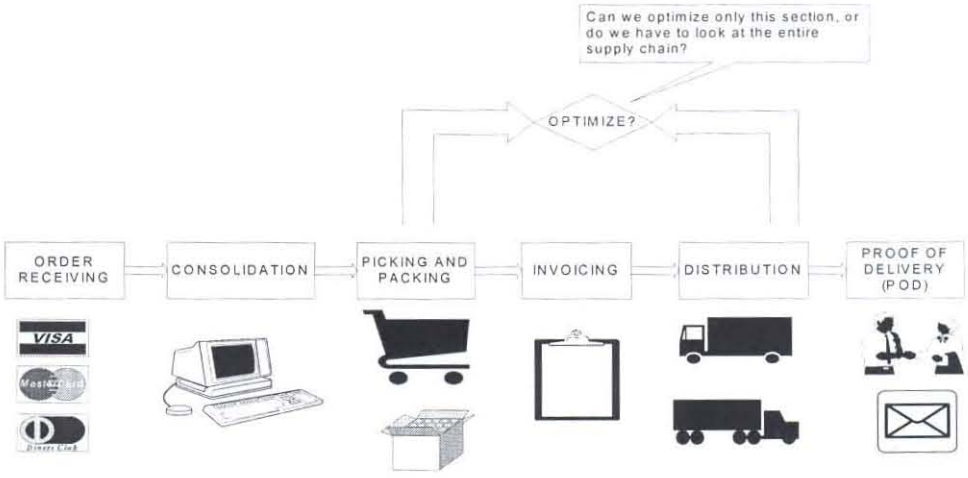

Figure 1: What part of the supply chain to optimize

The above diagram (Figure 1) depicts a typical supply chain from order receiving through to proof of delivery. The idea behind the development of this order fulfilment system is to optimise the supply chain from picking and packing through to distribution. Many packages focus entirely on the distribution side without looking at other affecting factors like picking and packing.

It is evident that there are at present a number of shortfalls within the fulfilment environment. There is thus scope for an effective, all-encompassing order fulfilment engine that addresses all problems, and ensures that all the positive aspects are maintained.

What is needed is an "order fulfilment engine" that provides companies with the tools they need to get the correct product to the correct place, at the correct time and cost, in the most optimal way – for every order and every customer. In order to develop such a fulfilment engine specific market requirements and information are essential.

1.2. Application Service Provider

There exist a need in the industry for an affordable service, which can assist with the optimization of distribution routes. Not all businesses have a large enough fleet to verify the costs associated with a fleet management system that includes optimization. Such a system normally requires a skilled operator, that add to the cost. A solution to this problem is the implementation of an optimization server in public domain. This is done by implementing a routing engine on an Application Service Provider (ASP). The ASP is a web-enabled distribution and fulfilment planning and optimization system that assist its users in the fulfilment of their customers' orders. This allows the provider to manage the system from one centralised server that allows other users to access the system via the Internet. This approach reduces infrastructure costs and speeds up the process as a powerful server instead of the computers of individual users manages the system.

Implementing software that enables you to optimize vehicle routes can result in major cost savings for a company. Unfortunately the costs associated with implementing such a system prevent companies to take this step. The

environment we are proposing is an e-fulfilment engine that is a web-enabled distribution and fulfilment planning and optimization system that is hosted in the form of an ASP (Application Service Provider). This means that a company no longer hosts and maintains its own computer software, but access the system via the Internet in real-time.

The implementation of the system on the ASP results in huge saving in terms of capital-, operational- and maintenance costs. To implement a sufficient solution, the system must incorporate

- **An effective optimization engine**. This will ensure that the client receives useful results. The engine is the heart of the system and is the topic of our discussion. The requirements of the engine are motivated from the implementation method, i.e. the engine must be able to handle different scenarios because it is located in an ASP environment, which is in essence a multi-user environment.
- **The ability to handle multiple clients**. This is the goal of the system, to provide cost effective solutions to clients that cannot afford the capital layout required to implement such a system. Each client has its own set of customers and depots, which the system cannot predict. It is therefore important for the engine to be robust and effective across different input scenarios.
- **Geographic locations of the customers**. This has a cost advantage for the clients because they do not need to keep geographic data on their systems. They can benefit from experts as well as additional data that will allow for professional maps as output. They can also benefit from an up to date road network on which the optimization is done. The service provider can ensure that the network used in the routing is representing the current status in the road network, e.g. peak and

off-peak travel times, road segments that is closed due to accidents or maintenance, as well as the addition of new segments. The optimization engine operates from a time and distance matrix between customers, which imply that these matrices must be up to date with the network and the engine must be able to handle multiple time matrices.

- **A management console**. To allow the user to specify certain parameters. These parameters can include the maximum route length or time, the open and close times of the depots, etc. The optimization engine must be able to enforce these constraints.

- **An easy operation interface**. This is not in the scope of this discussion, but form part of a successful ASP implementation. This include the ability to upload and download data to and from the system, which consists of customers with their order detail that must be uploaded from the client, and routes in the form of reports or maps that must be downloaded to the client.

Implementing the VRP with additional constraints has been defined as a complex problem. Implementing the VRP with additional constraints in an environment as described above, adds to the already complex problem. The algorithm cannot be designed to function well in one specific known environment, but must be able to adapt to the environment as specified by the client. This environment, or characteristics thereof, is not known at implementation time and the algorithm must be able to produce good results independent of the specific environment. The designed algorithm must be able to perform stable and reliable under these conditions, as well as producing acceptable results.

1.3. VRP and its Origin.

Vehicle Routing Problems (VRP) are an extension to the well-known Travelling Salesperson Problem. A number of visits being given, the goal is to perform these visits with vehicles, using a set of minimal-cost tours, each of which must start and end at the same position. The VRP is like the TSP, an NP-hard problem. If no extra constraints on the capacity or vehicles is given and a maximum driving time or tour length is not given, the solution to a VRP would be a single tour. However, real-life VRP comes precisely with these kinds of constraints, or even more complex ones.

Vehicle routing problems are all around as in the sense that many consumer products such as soft drinks, beer, bread, gasoline and pharmaceuticals are delivered to retail outlets by a fleet of trucks whose operations fits the vehicle routing model. In practice, the VRP has been recognized as one of the great success stories of operations research and it has been studied widely since the late fifties. Public services can also take advantage of these systems in order to improve their logistics chain. Garbage collection, or town cleaning, takes an ever-increasing part of the budget of local authorities.

The VRP was introduced by Dantzig and Ramser (1959) more than four decades ago. There has been since then a steady evolution in the design of solution methodologies, both exact and approximate, for this problem. The VRP is an NP-hard problem that is exceedingly difficult to solve to optimality. Yet, no known exact algorithm is capable of consistently solving to optimality instances involving more than 50 customers[2] and often requires relative few side constraints.

[2] Source: Golden et al., 1998; Naddef and Rinaldi, 2002 in [14], p. 3

Besides being one of the most important problems of operations research in practical terms, the vehicle routing problem is also one of the most difficult problems to solve. It is quite close to one of the most famous combinatorial optimization problems, the Travelling Salesperson Problem (TSP), where only one person has to visit all the customers. The TSP is an NP-hard problem. It is believed that one may never find a computational technique that will guarantee optimal solutions to larger instances for such problems. The vehicle routing problem is even more complicated. Even for small fleet sizes and a moderate number of transportation requests, the planning task is highly complex. Hence, it is not surprising that human planners soon get overwhelmed, and must turn to simple, local rules for vehicle routing.

In the m-TSP problem, m salesmen has to cover the cities given. Each city must be visited by exactly one salesman. All salesmen start from the same city (the depot) and must end their journey in this city again. We now want to minimize the sum of distances of the routes. The VRP is the m-TSP where a demand is associated with each city, and each salesmen/vehicles has a certain capacity (not necessarily identical). The sum of demands on a route cannot exceed the capacity of the vehicle assigned to this route. As in the m-TSP we want to minimise the sum of distances of the routes. Note that the VRP is not purely geographic since the demand may be constraining. The VRP is the basic model for a large number of different vehicle routing problems.

Many new side constraints have been added to meet real life needs. If we add a time window to each customer in the VRP we get the vehicle routing problem with time windows. In addition to the capacity constraint, a vehicle now has to visit the customer within a certain time frame. The vehicle may arrive before the time window opens. It is not allowed to arrive after the time window has closed.

Some models allowed for early or late servicing but with some form of additional cost or penalty.

1.4. Success of automated methods

Researchers often use models exhibiting some, but not all of the characteristics of real-world problems in order to test and evaluate their ideas.

The Vehicle Routing Problem (VRP) is no exception. The class of Vehicle Routing Problems is an intensive research area because of its usefulness to the logistics and transportation industry. For distribution companies, the transportation cost is the perfect target. Toth and Vigo (2002) (in Cordeau and Laporte [14], (p. 3)) report that the use of computerized methods in distribution processes often results in savings ranging from 5% to 20% in transportation costs. It is estimated that distribution costs account for almost half of the total logistics costs and in some industries, such as in the food and drink business; distribution costs can account for up to 70% of the value added costs of goods. This share has experienced a steady increase, since smaller, faster, more frequent, more on time shipments are required as a result of trends such as increased variability in consumer's demands, quest for total quality management, near-zero inventory production and distribution systems, sharp global-size competition.

1.5. Problem Environment

Knowledge of the problem environment can assist in developing a more effective algorithm. The problem environment consists of the constraints imposed on the problem, the input data that we have to work with and the objective function to minimise on.

This thesis considers a set of additional constraints added to the basic VRP. Although the ASP environment allow flexibility for the client to use these

constraints or not, the design does implement a different method for each possible combination of constraints. The design treats the omission of a constraint as a simple implementation of the constraint, e.g. if the client does not have heterogeneous fleet, the scenario can still be executed because homogeneous fleet is a subset of heterogeneous fleet. Thus a client with homogeneous fleet has a special case of the heterogeneous fleet problem. A client with single time windows has a special case of the multiple time windows problem. This implementation of a solution for an ASP ready algorithm will not include pre-processing of data to determine such special cases, but the guidance algorithm will handle the effectiveness of the algorithm.

Working in the ASP environment results in an unpredictable data environment. The input can differ from client to client. The objective of this study is to develop a solution that can operate in such surroundings. The thesis will provide a method to solve the problem efficiently, and is the first step towards providing a solution in the ASP environment that is flexible enough to provide a feasible solution. Although the primary goal of the ASP is to provide an affordable solution to the South African market, we cannot limit the input data efficiently. We can define the following basic scenarios:

- Short hauls with time window complexity –
- Short hauls with weight restriction
- Long hauls with time window complexity
- Long hauls with weight restriction
- Random located stops
- Cluster located stops

We must also take into account the driving conditions between the stops. The goal is to provide the algorithm with as much as possible data that simulates the practical environment. We simulate the travelling between stops with different travel times depending on the time of day, i.e. simulating peak and off-peak travel times.

1.6. Summary

Practical Vehicle Routing Problems come with additional constraints; for example, multiple capacity constraints can be expressed in several units and dimensions (weight, volume, length, number of pellets, etc.). Some problems involved constraints where the total capacity of the vehicle cannot be used; instead, the loading after vehicle must follow specific rules or legislation. This is for example the case in Europe with oil tanks.

The Application Service Provider environment allows the sharing of data and utilities via the Internet. This results in a cost-effective way to implement utilities that require specialized data and procedures. For a routing engine to function in this environment, it must be stable and flexible, and be able to handle the diversity of requests from multiple clients.

Chapter 2

2 PROBLEM BACKGROUND: VRP WITH MULTIPLE CONSTRAINTS

2.1. The Vehicle Routing Problem

Logistics can be defined as the provision of goods and services from a supply point to various demand points. The transportation of raw materials from the suppliers to the factory, from the factory to the depots, and the distribution to customers can be described as a complete logistic system. With an effective logistic system, cost can be reduced due to less penalties for late delivery, lowered trucking cost, shorter distances and effective use of capacity of the vehicle. One of the most significant measures of a logistic system is effective vehicle routing. Optimising of routes is the basis of vehicle routing problems.

The VRP originated from the Travelling Salesmen Problem (TSP). According to Winston [53] (p. 519) the TSP can be define as a problem where a salesperson must visit each of ten cities once before returning to his home. The cities need to be selected to minimise the total distance the salesmen travels.

According to Barbarosoglu et al. [3] (p. 256) the VRP can be described as the problem of designing optimal delivery or collection of routes from one or several depots to a number of customers subject to side constraints. Thus, the basic VRP can be described as vehicles that depart from the depot, visit one or more customers and return to the depot.

The VRP has a finite number of feasible solutions. The VRP solution space increase exponentially as the number of customers increases. Thus the VRP is known as a non-polynomial hard (NP-hard) problem.

The basic VRP is today no more than a classical problem. The advance of science has prompted the industry to ask for more real life solutions The basic VRP is given by a set of identical vehicles, a depot, a set of customers to be visited and a directed network connecting the depot and customers. Let us assume there are K vehicles, V = {0,1,2,3,...K-1}, and N+1 customers, C = {0,1,2,3,...N}. We denote the depot as customer 0, or C_0. Each arc in the network corresponds to a connection between two nodes. A route is defined as starting from the depot, going through a number of customers and ending at the depot. A cost c_{ij} and a travel time t_{ij} are associated with each arc of the network.

The problem is to find tours for the vehicles in such a way that:

- The objective function is minimized. The objective function can be the total travel distance, the number of vehicles used, or any cost related function.

Several constraints must be applied on the **basic** VRP:

- Only one vehicle handles the deliveries for a given customer. We will not split deliveries across multiple vehicles. A customer can only be visited once a day.

- The number of vehicles is equal to the number of routes, meaning that a vehicle can only complete one route per day.

The VRP has a finite number of feasible solutions. The VRP solution space increase exponentially as the number of customers increases. Thus the VRP is known as a non-polynomial hard (NP-hard) problem.

The basic VRP is today no more than a classical problem. The advance of science has prompted the industry to ask for more real life solutions The basic VRP is given by a set of identical vehicles, a depot, a set of customers to be visited and a directed network connecting the depot and customers. Let us assume there are K vehicles, V = {0,1,2,3,...K-1}, and N+1 customers, C = {0,1,2,3,...N}. We denote the depot as customer 0, or C_0. Each arc in the network corresponds to a connection between two nodes. A route is defined as starting from the depot, going through a number of customers and ending at the depot. A cost c_{ij} and a travel time t_{ij} are associated with each arc of the network.

The problem is to find tours for the vehicles in such a way that:

- The objective function is minimized. The objective function can be the total travel distance, the number of vehicles used, or any cost related function.

Several constraints must be applied on the **basic** VRP:

- Only one vehicle handles the deliveries for a given customer. We will not split deliveries across multiple vehicles. A customer can only be visited once a day.

- The number of vehicles is equal to the number of routes, meaning that a vehicle can only complete one route per day.

- The demand of the customers on every route is known with certainty. The demand of the customers in total on one route cannot exceed the capacity of the specific vehicle that will cover that route.

- The travelling distance between customer *i* and *j* are the same as the travel distance between *j* and *i*.

- The vehicles have the same capacity with the same fixed and variable cost, thus a homogeneous fleet are assumed.

- The vehicles must complete their route within a maximum length of time, usually the time the depot is open.

- The vehicle returns to the depot at the end of the route.

The VRP can be formulated as follows:

- 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, $V = \{0, 1, 2, \ldots, K - 1\}$, and $N + 1$ customers, $C = \{0, 1, 2, \ldots, N\}$.

- For simplicity, we denote the depot as customer 0.

- Each arc in the network corresponds to a connection between two nodes.

- A route is defined as starting from the depot, going to any number of customers and ending at the depot.

- The number of routes in the traffic network is equal to the number of vehicles used, 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, 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 travelled by vehicle k.

- 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 a VRP.

- The decision variable $x_{ijk}, (i, j = 0,1,2..N; k = 0,1,2..K; i \neq j)$ 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. $c_{ij} < c_{ih} + c_{hj}$ and $t_{ij} \leq t_{ih} + t_{hj} \forall h, i, j \in N$ need not apply.

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:

Notation:

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.

m_i = demand at node i.

q_k = capacity of vehicle k.

e_i = open time at node i.

l_i = close time at node i

t_i = arrival time at node i.

f_i = service time at node i.

r_k = maximum route time allowed for vehicle k.

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 travelled 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.

Principle decision variable: $x_{ijk} = \{0,1\}$: 0 if there is no arc between node i and j and 1 otherwise.

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

Subject to:

$$\sum_{k=0}^{K-1} \sum_{j=1}^{N} x_{ijk} = K \ \ for \ i = 0 \qquad (2)$$

$$\sum_{j=1}^{N} x_{ijk} = \sum_{j=1}^{N} x_{jik} \leq 1 \quad for\ i = 0; k \in [0, K-1] \tag{3}$$

$$\sum_{k=0}^{K-1} \sum_{j=1}^{N} x_{ijk} = 1 \quad for\ i = 1,2..N \tag{4}$$

$$\sum_{i=0, i \neq h}^{N} x_{ihk} - \sum_{j=1, j \neq h}^{N} x_{hjk} = 0 \quad \forall h \in [1, N]; k \in [0, K-1] \tag{5}$$

$$u_i - u_j + N x_{ij} \leq N - 1 \quad for\ i \in [1, N]; j \in [1, N]; i \neq j \tag{6}$$

$$\sum_{i=0}^{N} m_i \sum_{j=0, j \neq i}^{N} x_{ijk} \leq q_k \quad \forall k \in [0, K-1] \tag{7}$$

$$\sum_{i=0}^{N} \sum_{j=0, j \neq i}^{N} x_{ijk} (t_{ij} + f_i + w_i) \leq r_k \quad \forall k \in [0, K-1] \tag{8}$$

- The objective function of the problem is given in (1).
- Constraint (2) specifies that there are exactly K routes going out of the depot.
- The third constraint (3) makes sure that each route leaves the depot and return to the depot
- Constraints (4) and (5) make sure exactly one vehicle goes to and leaves a customer.
- Constraint (6) ensures that there are no sub-tours in the solution. A sub-tour is a route that does not pass through the depot.

- (7) is the capacity constraint.

- Maximum travel time for each vehicle is assured in Eq. (8).

The model described in this section is a standard mathematical model for a basic VRP problem. When additional constraints are needed, they must be added to the existing constraints in the model or some of the existing constraints must be relaxed.

The industry requires additional constraints on the basic VRP. Additional constraints that we will address include:

- The limitation of the length, duration or cost of each individual tour. This restricts a route for running too long, which can result in overtime costs, insufficient fuel, etc.

- The addition of a service time for each customer. The volume of the stock to be delivered can have an influence on the service time at a customer. The delivery time will have an influence on the total route time and must be taken into account.

- The addition of time windows during which the customers have to be visited. The problem we will discuss is the use of multiple time windows, i.e. the customer can specify more than one time period available for delivery.

- The vehicle can return to the depot and have enough time for another route before the maximum allowed time is up. This will allow double scheduling, which will result in a cost saving, as the second route utilize the same vehicle and reduce the number of vehicles required to service all the customers.

- The travel time can vary between customers depending on the time of day. This implies peak and off-peak travel times.

- The fleet is not necessarily homogeneous, i.e. vehicles can differ in capacity and cost. This might result in a good solution to use the vehicles with a large capacity to pick up customers that is far away form the depot.

- A vehicle can have a specified available time. This allows for certain vehicles to be out in the field longer to cater for long routes. The implementation will add time window constraints to a vehicle.

We need to redefine the mathematical model for our problem. We will make use of the base model with the following changes:

- Constraint (2) is now invalid and will be replace by

$$\sum_{j=1}^{N} x_{ijk} \leq p_k \quad for\ i = 0; k \in [0, K-1] \tag{2}$$

where p_k is the maximum number of routes allowed for vehicle k.

The number of routes going out of the depot for a specific vehicle are constrained to a maximum of p_k , which implies that a vehicle can now have multiple routes done in a day.

- We impose time windows at a stop

$$t_0 = 0 \tag{9}$$

$$t_i + x_{ijk}(t_{ij} + f_i + w_i) \leq t_j \quad i, j \in [1, N]; i \neq j; k \in [0, K-1] \tag{10}$$

$$e_i \leq t_i \leq l_i \tag{11}$$

- We redefine the service time at each stop as

  f_i = Fixed Time + (Variable Time * m_i)

- We also redefine the meaning of travel time

  t_{ij} = Travel Time at $(t_i + f_i + w_i)$

  which calculates the travel time from i to j depending on the departure time at i.

- We just make a note that q_k is not necessarily the same for each vehicle.

- The monetary cost of a route can be calculated as follows

$$C(R_{ki}) = (F_k / \sum_{j=1}^{N} x_{ijk}) + (D_k * V_k) \quad for\ i = 0; k \in [0, K-1]$$

  where the first term is the fixed cost of the vehicle divided into the number of routes and the second term is the distance of the route multiplied by the running cost of the vehicle.

2.2. Meta Heuristics

The implementation of an algorithm that can efficiently and in reasonable time solves the aforementioned problem has not been successfully implemented before. To embark on a journey to find a sufficient algorithm requires investigation of existing problems and solutions as well as inventing new methods. Several papers have been presented that solve the VRP with additional side constraints. They mainly focus on solving the basic VRP with one or two

additional side constraints. Some of the most popular problems include the VRP with time windows and the VRP with pickup and delivery.

Heuristic methods play an important role in solving problems with this complexity. Most solutions include a heuristic method, or a hybrid of heuristic methods at the heart of the solution. In the next section, we will discuss some of the more popular heuristic methods.

Meta-heuristics, or global optimization heuristics, have a common feature: they guide a subordinate heuristic in accordance with a concept derived from artificial intelligence, biology, nature or physics to improve their performance.

Meta-heuristics succeed in leaving the local optimum by temporarily accepting exchanges that decrease the objective function value. Meta-heuristics use information of the problem environment and the nature of the objective function to direct the search process to regions that promise better solutions. It is possible that the meta-heuristic will return to the local optimum without finding a better solution. This is called cycling and can be avoided by adjusting the heuristic's settings to allow more degrading moves for longer.

The concept of a heuristic being trapped at a local optimum can be demonstrated in Figure 2. If a heuristic finds a solution S, with objective function F(S), where S is close to point C, then it will only improve until it gets to local optimum C. No further improvements in the objective function will be achievable, because all moves will reduce the objective function. However, if a meta-heuristic finds a solution close to point B, degrading moves will be allowed that may direct the search to the global optimum, point A.

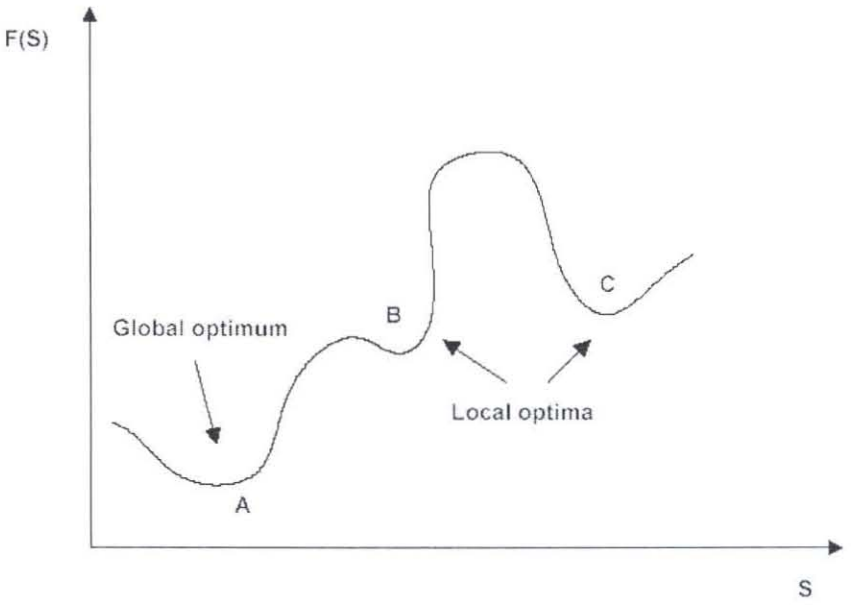

Figure 2: Global and Local Optima

Meta-heuristics will be successful on a given optimisation problem if it can provide a balance between the exploitation of the accumulated search experience and the exploration of the search space to identify regions with high quality solutions in a problem specific, near optimal way. The various meta-heuristics are classified according to the following criteria:

- **Trajectory methods vs. discontinuous methods**: Trajectory methods like SA and TS follow one single search trajectory corresponding to a closed walk on the neighbourhood graph. Discontinuous methods allows larger jump in the neighbourhood graph.

- **Populated-based vs. single-point search**: In single-point search only one single solution is manipulated at each iteration of the algorithm. TS

and SA are single-point search methods. GA and ant colony algorithms is Population-based.

- **Memory usage vs. memoryless methods**: Meta-heuristics with memory are the TS, GA, SS and ant systems. According to Taillard et al. [48] these meta-heuristics with memory can be viewed as adaptive memory programming (AMP) heuristics. The term "memory" was used explicitly for TS, but other meta-heuristics use mechanisms that can be considered as memories. There are meta-heuristics that cannot be entered into the AMP methods, such as SA. However they may be included in the improvement procedure of AMP.

- **One vs. various neighbourhood structures**: SA and TS algorithms are based on one single neighbourhood structure. Other algorithms such as Iterated Local Search typically use at least two different neighbourhood structures.

- **Dynamic vs. static objective function**: Some algorithms modify the evaluation of the single search states during the run of the algorithm. In the use of a dynamic objective function penalties for the inclusion of certain solution attributes that modify the objective function are introduced. TS may be interpreted as using dynamic objective function, as some point in the search is forbidden, corresponding to infinitely high objective function values. The other algorithms use static objective functions.

Evaluation of heuristic methods consists of comparing criteria such as running time, quality of solution, ease of implementation, flexibility and robustness. For the purpose of our algorithm, flexibility is an important consideration. The algorithm should be able to handle changes in the data patterns, side constraints

and objective function, as each client has his own specific requirement. We are not working on a predetermined set of data with a specified objective function. Working in such an environment make it possible to find a method that is effective for that specific environment by making use of the knowledge about the problem.

Because the heuristic methods are non deterministic, i.e. we cannot predict the result even if we apply the same algorithm on the same data with the same number of iterations, the algorithm should not perform poorly on any instance, as well as being able to produce a good solution each time it is applied to the same instance.

We will also try to validate the applicability of the method on our problem by discussing the design of the method is well as what we see as its advantages and disadvantages. With this approach we will filter out certain methods. Comparisons discussed in this paper are from existing papers, which mainly present the best results found for the method. Comparison is also made difficult because solutions were not all implemented on the same computer (running time), and have not all use the same number of iterations. Existing methods is also not designed for our specific problem and thus we cannot really compare methods outright to decide on a method to implement for our problem.

Using only the best results of a non-deterministic heuristic, as is often done in the literature, may create a false picture of its real performance. We considered average results based on multiple executions on each problem an important basis for the comparison of non-deterministic methods. Furthermore, it would also be important to report the worst-case performance.

Moreover, an algorithm should be able to produce good solutions every time it is applied to a given instance. This is to be highlighted since any heuristics are non-

deterministic, and contain some random components such as randomly chosen parameter values. The output of separate executions of these non-deterministic methods on the same problem is in practice never the same. This makes it difficult to analyze and compare results.

Heuristic methods

An algorithm is said to be efficient when it runs in polynomial time, i.e., its running time is not longer than a polynomial function of the size of the problem. An algorithm is said to be effective if it produces high-quality solutions, preferably in less time than any efficient algorithm for the problem. The most preferred algorithms are both efficient and effective. If the algorithm produces the mathematically best solution it is called optimal (or exact) if it produces a good but not necessarily best solution it is called heuristic. A construction algorithm constructs a solution to a problem, whereas an improvement algorithm works on an existing solution to obtain better levels performance measures.

According to Laporte [33], heuristics belong to two broad classes: classical heuristics and modern heuristics (or metaheuristics). Classical heuristics can be broadly classified into three categories. Constructive heuristics gradually build a feasible solution while keeping an eye on solution cost, but do not contain an improvement phase per se. In two-phase heuristics, the problem is decomposed into its two natural components: clustering of vertices into feasible routes and actual route construction, with possible feedback loops between the two stages. Two-phase heuristics can be divided into two classes: cluster-first, route-second methods and route-first, cluster-second methods. In the first case, vertices are first organized into feasible clusters, and a vehicle route is constructed for each of them. In the second case, a tour is first built on all vertices and is then segmented into feasible vehicle routes. Finally, improvement methods attempt to upgrade any feasible solution by performing a sequence of edge or vertex exchanges

within or between vehicle routes. The distinction between constructive and improvements methods is, however, often blurred since most constructive algorithms incorporate improvements steps at various stages.

As far as we are aware, six main types of metaheuristics have been applied to the VRP:

1) Simulated Annealing (SA),

2) Deterministic Annealing (DA),

3) Tabu Search (TS),

4) Genetic Algorithms (GA),

5) Ant Systems (AS), and

6) Neural Networks (NN).

The first three algorithms, SA, DA and TS, start from an initial solution x_1 , and move at each iteration t from x_t to a solution x_t+1 in the neighborhood $N(x_t)$ of x_t, until a stopping condition is satisfied. If $f(x)$ denotes the cost of x, then $f(x_t+1)$ is not necessarily less than $f(x_t)$. As a result, care must be taken to avoid cycling. Put paragraph in bullets

GA examines at each step a population of solutions. Each population is derived from the preceding one by combining its best elements and discarding the worst. AS is a constructive approach in which several new solutions are created at each iteration using some of the information gathered at previous iterations. As was pointed out by Taillard et al. [48], TS, GA and AS are methods that record, as the search proceeds, information on solutions encountered and use it to obtain improved solution. NN is a learning mechanism that gradually adjusts a set of

weights until an acceptable solution is reached. The rules governing the search differ in each case and these must also be tailored to the shape of the problem at hand. Also, a fair amount of creativity and experimentation is required.

The following sections discuss the most applicable methods.

2.2.1. Simulated Annealing (SA)

Simulated Annealing searches the solution space by simulating the annealing process in metallurgy (Qili et al [39]). The algorithm jumps to distant location in the search space initially. The size of the jumps reduces as time goes on or as the temperature "cools" down. Eventually the process will turn into local search descent method.

One of its characteristics is that for very high temperatures, each state has almost equal change to be the current state. At low temperatures only states with low energy have a high probability of being the current state. These probabilities are derived for a never ending executing of the metropolis loop. The advantages of the scheme is:

- 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.

However this original version from SA has some drawbacks

- Repeated annealing with a 1/log k schedule is very slow, especially if the cost function is expensive to compute, which will be the case for our problem.

- For problems where the energy landscape is smooth, or there are few local minima, SA is an overkill – simpler faster methods works 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 if it has found and optimal solution.

2.2.2. Tabu Search (TS)

The word Tabu (or taboo) comes from Tongan, a language of Polynesia, where it was used by the aborigines of Tonga Island to indicate things that cannot be touched because they are sacred.[3] According to Webster's Dictionary, the word now also means "a prohibition imposed by social custom as a protective measure" or of something "banned as constituting a risk." These current more pragmatic senses of the word accord well with the theme of Tabu search. The risk to be avoided in this case is that of following a counter-productive course, including one, which may lead to entrapment without hope of escape. On the other hand, as in the broader social context where "protective prohibitions" are capable of being superseded when the occasion demands, the "taboos" of Tabu search are to be overruled when evidence of a preferred alternative becomes compelling.

[3] Source: Tabu Search Network [31]

Tabu Search (TS) is a local search metaheuristic introduced by Glover (1986). TS explores the solution space by moving at each iteration from a solution to the best solution in a subset of its neighbourhood N(s). Contrary to classical descent methods, the current solution may deteriorate from one iteration to the next. Thus, to avoid cycling, solutions possessing some attributes of recently explored solutions are temporarily declared Tabu or forbidden. The duration that an attribute remains Tabu is called its Tabu-tenure and it can vary over different intervals of time. The Tabu status can be overridden if certain conditions are met; this is called the aspiration criterion and it happens, for example, when a Tabu solution is better than any previously seen solution. Finally, various techniques are often employed to diversify or to intensify the search process.

The most important association with traditional usage, however, stems from the fact that taboos as normally conceived are transmitted by means of a social memory, which is subject to modification over time. This creates the fundamental link to the meaning of "taboo" in Tabu search. The forbidden elements of Tabu search receive their status by reliance on an evolving memory, which allows this status to shift according to time and circumstance.

TS is the only metaheuristic that has been explicitly developed with a memory. In a sense this method imitates the human being looking for a good solution of a combinatorial optimization problem. Glover proposed a number of strategies to guide the search and make it more efficient. TS is open for any strategy well adapted to the problem on which it is applied.

More particularly, Tabu search is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the

search, TS contrasts with memoryless designs that heavily rely on semi random processes that implement a form of sampling. Examples of memoryless methods include semi greedy heuristics and the prominent "genetic" and "annealing" approaches inspired by metaphors of physics and biology. Adaptive memory also contrasts with rigid memory designs typical of branch and bound strategies. (It can be argued that some types of evolutionary procedures that operate by combining solutions, such as genetic algorithms, embody a form of implicit memory. However, this form of memory is not sufficient to embrace many aspects of what we normally conceive to be a hallmark of 'intelligent' problem solving. Tabu search also has implicit memory features that offer opportunities for establishing more effective variants of evolutionary approaches.)

The emphasis on responsive exploration in Tabu search, whether in a deterministic or probabilistic implementation, derives from the supposition that a bad strategic choice can yield more information than a good random choice. In a system that uses memory, a bad choice based on strategy can provide useful clues about how the strategy may profitably be changed. (Even in a space with significant randomness a purposeful design can be more adept at uncovering the imprint of structure.)

Responsive exploration integrates the basic principles of intelligent search, i.e., exploiting good solution features while exploring new promising regions. Tabu search is concerned with finding new and more effective ways of taking advantage of the mechanisms associated with both adaptive memory and responsive exploration. The development of new designs and strategic mixes makes TS a fertile area for research and empirical study.

The main advantage of the basic version is its aggressiveness: the search converges toward the local optimum and examines the neighbourhood of this local optimum very quickly. However, it can easily get trapped in a sub-space

containing only solutions of poor quality. To diversify the search and force it to visit solutions with different characteristics, one basic idea was to increase the number of forbidden components when performing local modifications to a solution. So, the discussion quickly turned around the optimum tabu list size, since the short list allows a thorough examination of the neighbourhood of a good solution while a long list facilitates the escape from a local optimum to explore new regions of the search space. The reactive Tabu search proposed by Battiti and Tecchiolli (1994) (in Bräysy [5], p. 4) was designed to automatically adapt the Tabu list size and avoid the fastidious task of explicitly managing the Tabu list.

The main difficulty with TS is thus to efficiently incorporate diversification and intensification mechanisms. The use of a memory that stores good solutions visited during the search and the design of a procedure to create provisional solutions from it is a way to achieve this goal. Indeed, solutions contained in memory during the initial search phase present different characteristics, thus leading to a diversified search. Later, solutions contained in memory are mostly representative of one or a few good regions of a solution space. The result is that the search gradually shifts from diversification to intensification.

2.2.3. Genetic Algorithms (GA)

The Genetic Algorithm (GA) is an adaptive heuristic search method based on population genetics. The basic concepts were developed by Holland (1975) (in Ombuki et al, [39], p.3), while the practicality of using the GA to solve complex problems was demonstrated in De Jong (1975) and Goldberg (1989) (in Bräysy and Gendreau, [8], p. 10).

GA evolves a population of individuals encoded as chromosomes by creating new generations of offspring through an iterative process until some convergence

criteria are met. Such criteria might, for instance, refer to a maximum number of generations, or the convergence to a homogeneous population composed of similar individuals. The best chromosome generated is then decoded, providing the corresponding solution.

The creation of a new generation of individuals involves three major steps or phases: selection, recombination and mutation. The selection phase consist of randomly choosing two parent individuals from the population for mating purposes. The probability of selecting a population member is generally proportional to its fitness in order to emphasize genetic quality while maintaining genetic diversity. Here, fitness refers to a measure of profit, utility or goodness to be maximized while exploring the solution space. The recombination or reproduction process makes use of genes of selected parents to produce offspring that will form the next generation. As for mutation, it consists of randomly modifying some gene(s) of a single individual at a time to further explore the solution space and ensure, or preserve, genetic diversity. The occurrence of mutation is generally associated with a low probability. A new generation is created by repeating the selection, reproduction and mutation processes until all chromosomes in the new population replace those from the old one. A proper balance between genetic quality and diversity is therefore required within the population in order to support efficient search.

Although theoretical results that characterize the behaviour of the GA have been obtained for bit-string chromosomes, not all problems lend themselves easily to this representation. This is the case, in particular, for sequencing problems, such as the vehicle routing problem, where an integer representation is more often appropriate. Therefore, in most applications to VRPTW, the genetic operators are applied directly to solutions, represented as integer strings, thus avoiding

coding issues. In most cases the authors use delimiters to separate customers served by different routes.

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 determine the performance of its application: representation of solution, initial population, selection, reproduction, and population improvements (Qili, [39], p. 72).

Advantages

- GA is very flexible with a lot of parameters to adjust for different needs;
- GA generally explores a larger neighbour 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.

Disadvantages

- 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 at enough.

Coding a solution with a binary vector is not natural and can significantly impact the performance. Hence, binary coding was replaced by a more natural representation of solutions. The classical cross over operators does not correspond to logical operations on solutions. Furthermore, the use of other representations and binary vectors naturally led to the design of specialized operators, well adapted to the solution representation and capable of generating new feasible solutions. GA can easily identify different solution sub spaces with good characteristics, but they lack the "killer instinct" that would allow them to intensify the search into these areas. To alleviate this weakness, the mutation operation was replaced by repair procedures and local search.

2.2.4. Ant Systems (AS)

The idea of imitating the behaviour of ants to find solutions to combinatorial optimization problems was initiated by Colorni, Dorigo and Maniezzo (in Bullnheimer et al, [12], p. 1). The metaphor comes from the way ants search for food and find a way back to the nest. Initially ants explore the area surrounding their nest in a random manner. As soon as an ant finds a source of food, it evaluates the interest of the source (quantity and quality) and carries some of food to the nest. During the return trip, the ant leaves on the ground a chemical pheromone trail whose quantity depends on the quality of the source. The role of this pheromone trail is to guide other ants toward the source. After a while, the path to a good source of food will be indicated by a large pheromone trial, as a trial grows with number of ants that reach the source. Since source is that are close to the nest are visited more frequently than those that are far way, pheromone trials leading to the nearest sources grow faster. The final result of this process is that ants are able to optimize their work.

The transposition of this food searching the area into an algorithm framework for solving combinatorial optimization problems is octane through an analogy between:

- the search area of the real ants and the set of feasible solutions to the combinatorial problem;
- the amount of food associated with the source and the objective function;
- the pheromone trial and an adaptive memory.

The most important component of an ant system is the management of the pheromone trials. In a standard ant system, pheromone trials are used in conjunction with the objective function to guide the construction of new solutions. Once a solution has been produced, a standard ant system updates the pheromone trials as follows: first all trials are a weakened to simulate the evaporation of pheromone; then, pheromone trials that correspond to components that were used to construct the resulting solution are reinforced, taking into consideration the quality of this solution.

Based on the previous general scheme different AS implementations have been proposed where pheromone updating is performed in different ways. Different ways of modifying pheromone values generate different types of search mechanisms. Recently it has been shown that AS based algorithms are being powerful in combination with local search procedures. In these situations pheromone information is used to produce solutions (diversification phase) that are optimized by a local search (intensification phase). Optimize solutions are then used to update pheromone information and new solutions are successively generated by the ants.

Like GA, early implementations of the ant system converged too slowly toward high-quality solutions. Therefore, intensification mechanisms were gradually introduced. The most recent implementations incorporate local search mechanisms to improve the solutions produced by the ants.

2.3. Existing Methods and Implementations

The vehicle routing problem has many variants that have been attempted by many people with different criteria and different methods. The question arises on how could another study on the problem be feasible. In the following section we will discuss some of the existing implementations of the VRP. This section will discuss some implementations which will enable us to derive methods already tested, or show incompleteness in there implementation for our use. It must be noted that certain methods were not considered as feasible because it was deemed too slow. We can reconsider these methods because of the improvement in computing power in recent years.

Table 1 is a present state of work done of the study to derive a feasible solution for our problem. The model indicates the model implemented by the author that is of interest to us. The following section will discuss the methods in detail.

Present State			
Author	*Year*	*Problem*	*Model*
Amberg, Domschke and Voβ	2000	M-CARP	Cluster First Route Second
Taillard, Laporte and Gendreau	1995	VRPM	Tabu Search, generating and combining promising solutions.
Lau and Liang	2000	PDPTW	Two-staged heuristic, Construction and Tabu Search, working with job pairs
Salhi et al	1992	VFM	Unlimited vehicles, best vehicle selection
Taillard	1996	VRPHE	Column generation method
De Backer and Furnon	1997	VRPTW	Constraint programming, routestop has next stop
Xu and Kelly	1999	VRPTW	TS with independent tabu tenure per operation
Ombuki, Nakamura and Osamu	2002	VRPTW	Hybrid GA and TS
Van Schalkwyk	2002	VRPTW	Time Window Compatibility, selective neigbour list

Table 1: Present State

2.3.1. Multiple depot

Although we do not focus on a multiple depot implementation of the VRP, it is important to understand the methods available for solving this problem. In our problem we make use of the cluster first route second (CFRS) method. CFRS methods are more suitable for node routing problems. The clustering method is

left to the specific client, i.e. the nodes will be clustered with an algorithm selected by the client before we receive the data.

In the capacitated arc routing problem with multiple centres the objective is to find routes starting from the given depots or centres such that each required arc is served, capacity and usually additional constraints are satisfied and total travel cost is minimised. The paper of Amberg et al, [1] consider a heuristic transformation of the multiple centre arc routing problems into a multiple centre capacitated minimum spanning tree problem with arc constraints. Arc routing applications referred to problems where the distribution or collection of goods is bound up with traversing a distance such as mail delivery, snow removal, garbage disposal, street sweeping and police patrols. Thus, the customers are modelled as arc or edges, whereas in node routing problems the customers correspond with the nodes as, e.g. in the travelling salesman problem. The well-known Chinese postman problem (CPP) is the basic arc routing problem was named after the Chinese scientist Mei-Ko Kwan (1962) who was the first to publish on this problem.

Introducing additional constraints even in undirected or directed graphs usually yields NP-hard problems such as the capacitated Chinese postman problem, where the capacity of the postman is restricted, or the rural postman problem (RPP) where the set of required arcs (i.e. those arcs which need serving) need not be connected and has to be linked using non-required arcs. With respect to developing solution methods, it is important to note that capacitated arc routing problems consist of two interdependent sub problems: The assignment problem which forms subsets or clusters of required arcs served by the same vehicle and the sequencing or routing problem which determines the sequence of serving the arcs.

2.3.2. *Pickup and Delivery*

We consider existing pickup and delivery problems to determine the similarity between it and multiple routes per vehicle. The Pickup and Delivery Problem with Time Windows (PDPTW) models the situation in which a fleet of vehicles must service a collection of transportation requests. Each request specifies a pickup and delivery location. The multiple routes per vehicle problem can be seen as a pickup from the depot and delivery to the customer. The route can stop several times at the depot to pickup goods for more customers. The depot must now also have a service time. While VRPTW is well studied, there is relatively less literature on PDPTW. Moreover, no one has developed comprehensive benchmark PDPTW instances that facilitate experimentation of new approaches.

Lau and Liang [35] presented a two-staged method to solve the pickup and delivery problem with time windows (PDPTW). In the first phase, they apply a novel construction heuristics to generate an initial solution. In the second phase, a tabu search method is proposed to improve the solution. In their model, they assume there is an unlimited number of vehicles and all vehicles have the same capacity. Lau and Liang implement a partitioned insertion heuristic, which is a hybrid heuristic combining the advantages of the standard insertion heuristic and sweep heuristic. The stops are inserted into the route as pairs, ensuring that a pickup stop is always on same route as the delivery route. They introduce three different neighbourhood moves, namely, Single Pair Insertion (SPI), Swap Pairs between Routes (SBR) and Within Routes Insertion (WRI).

The study of this method indicates that the VRPTW was adapted to work in pairs. Implementing the VRPTW with multiple routes per vehicle is less complex than the PDPTW. This thesis presents a similar approach as was presented by Lau et al [35]. From the results of Lau et al [35] study we conclude that some minor changes to the operators in our problem would be sufficient to solve the

additional constraint of allowing multiple routes per vehicle. Where the PDPTW needs to check for pairs, we will be force to check the affect of route alterations on subsequent routes.

2.3.3. VRP with Multiple use of vehicles

The Vehicle routing problem with multiple use of vehicles is a variant of the standard vehicle routing problem in which the same vehicle may be assigned to several routes during a given planning period. Taillard et al, [49] presented a tabu search heuristic for this problem.

One drawback of the standard VRP definition is that it implicitly assumes each vehicle is used only once over a planning period of duration M. For example, M could correspond to an eight-hour working day. In several contexts, once the vehicle routes have been designed, it may be possible to assign several of them to the same vehicle and thus use fewer vehicles. When m is given a priori and Q is relatively small, this will often be the only practical option. However, this possibility is not directly accounted for in the problem statement and more often than not, an efficient "packing" of the routes into working days will be hard to achieve. Designing routes with multiple uses of the vehicles is rather important in practice, but this problem (denoted by the abbreviation VRPM) has received very little attention in the Operational Research literature.

In recent years, several powerful tabu search algorithms have been proposed for the VRP. As a rule, these algorithms produce very good and sometimes optimal solutions. Rochat and Taillard presented an algorithm that allows diversification of the search process to take place by generating and combining promising solutions, not unlike what is done in genetic algorithms. More precisely, the route generation procedure first produces several good VRP solutions using tabu search. It then extracts single vehicle routes from this population of solutions,

and combines some of these routes to define a partial starting solution for another application of tabu search. This process is repeated a number of times and some of the vehicle routes generated are selected as candidates for the final VRP solution. Note that each application of tabu search has the effect of producing a full VRP solution starting from a limited set of routes and it may also modify these seed routes through the local search process.

Taillard et al [49] proposed a heuristic for the VRPM based on the algorithm of Rochat and Taillard. The proposed heuristic is made up of three parts. It first generates a large set of good vehicle routes satisfying the VRP constraints. It then makes a selection of a subset of these routes using an enumerative algorithm. Finally, it assembles the selected routes into feasible working days using several applications of a bin packing heuristic.

2.3.4. Heterogeneous Fleet

We considered work done on heterogeneous fleet for obvious reasons. The vehicle routing problem with a heterogeneous fleet of vehicles (VRPHE) is a major optimization problem. Indeed, most companies that have to deliver or collect goods own a heterogeneous fleet of vehicles. We will not consider composition of vehicles, although it is relevant to some of the problems in the industry.

The problem of composition of vehicles includes the additional problem of deciding which trailer goes with which vehicle. We solve this problem by building a vehicle set beforehand, and checking the vehicle capacity after routing. If the capacity is enough for the vehicle alone, the trailer is left at home and the total route cost is reduced.

The VRPHE has attracted much less attention than the VRP or VRPTW. This is mainly due to the fact that the VRPHE is much harder to solve than the classical VRP. Taillard [46] propose a heuristic column generation method for the VRPHE.

Taillard [46] defines the heterogeneous fleet as follows: In the heterogeneous problems, we have a set $\Psi = \{1, \ldots, K\}$ of different vehicle types. A vehicle of type $k \in \psi$ has a carrying capacity Q_k. The number of vehicles of type k available is n_k. The cost of the travel from customer i to j $(i, j = 0, \ldots, n)$ with a vehicle of type k is d_{ijk}. The use of one vehicle of type k implies a fixed cost f_k. Our implementation defines a fleet in a similar way.

A special case of VRPHE is the fleet size and mix vehicle routing problem (Golden et al., 1984 in Taillard [46]) also called the fleet size and composition VRP or the vehicle fleet mix (VFM, Salhi et al., 1992 in Taillard [46]). The goal of this problem is to determine a fleet of vehicles such that the sum of fixed costs and travel costs is minimized. This problem is a particular VRPHE for which :

1) The travel costs are the same for all vehicle types $(d_{ijk} = d_{ijk'}, k, k' \in \psi)$.

2) The number n_k of vehicles of each type is not limited $(n_k = \infty, k \in \psi)$.

We view this kind of problem as a strategic optimization and it will not be considered. Our problem is more concerned with the current situation at the depot, i.e. the fleet is already there, we cannot make major alterations on the fleet, but we must still try and optimise the vehicle use as best as we can. If results continuously show that a certain vehicle is not necessary, it can be considered to remove the vehicle from the system and determine if the algorithm still returns feasible solutions.

Another special case of the VRPHE is the VFM with variable unit running costs (VFMVRC, Salhi et al., 1992 in Taillard [46]). The VFMVRC is a particular VRPHE for which $(n_k = \infty, k \in \psi)$. Several papers on the VFM have been published. Golden et al. (1984) were among the first to address this problem. This problem's goal is similar to the VFM and is also strategic. We will not consider this implementation. Much less work has been done for the VRPHE. Let us quote the taboo searches of Semet and Taillard (1993) and Rochat and Semet (1994) (in Taillard [46]) for real-life problems including many other constraints.

"For homogeneous VRPs, many heuristic methods have been proposed. Among the most efficient ones, are the adaptive memory procedure (AMP) of Rochat and Taillard (1995) and the taboo search of Taillard (1993). This last method uses a local search mechanism based on the move of one customer from one tour to another or the exchange of two customers that belong to different tours. Since the vehicles are identical, it is easy to check the feasibility of a move and to evaluate its cost. For the VRPHE, the feasibility check or the evaluation of a move requires finding a new assignment of the vehicles to the new solution's tours. In Semet and Taillard (1993), several techniques have been proposed to simplify and accelerate the re-assignment of vehicles to tours. However, the re-assignment problem is very simple in the case of the VFM: each tour is performed with the cheapest vehicle type that is able to carry all the orders of the tours. This is certainly a reason that the VFM has been more studied than the VRPHE."

The above quote is a warning on the addition of heterogeneous fleet to our VRP, especially if we do not apply it in the sense of the VFM. We will show, however, that the methods used in our implementation are sufficient enough and effective in a reasonable time period.

Taillard presents a heuristic column generation method for solving the VRPHE. The column generation is based on the AMP of Taillard (1994), which uses an embedded taboo search. Taillard proposes to treat the VRPHE by solving a succession of homogeneous VRPs, since the solution methods for homogeneous VRPs are becoming more and more efficient. For each type of vehicle, they solve a homogeneous VRP (without limitation on the number of vehicles available) with an AMP. The tours of the homogeneous VRP solutions are then combined to produce a solution to the VRPHE.

The AMP first generates a set of good solutions using the taboo search. It then extracts single vehicle tours from this set of solutions, and combines some of these tours to define a partial starting solution for another application of taboo search. This process is repeated a number of times and the tours are memorized as candidates for the final VRPHE solution. Once the homogeneous VRPs are solved for each vehicle type, one has a set T of tours that have been memorized. The useless tours of T are removed: only one copy of each tour is kept in T; the dominated tours are eliminated (a tour is dominated if it is more expansive than another tour of T servicing the same customers). In the case of the VFM, the algorithm always produces a feasible solution if the iterative search used to solve the homogeneous VRP succeeds in finding feasible solutions.

In our objective, the proposed solution is not considered for the following reasons:

- In the case of the VFM, an unlimited number of vehicles exist to solve the problem. Whatever feasible tour is selected from the homogeneous solution list is possible, as the vehicle exist. In our instance, it might happen that the selected vehicle route cannot be used, as the number of routes for the type of vehicle already equals the number of vehicles

available. Another vehicle must be selected for this route, which might not result in the best solution.

- If we start to make alterations to the selection of vehicles, it might happen that the routes in the list for a specific vehicle on a specific stop are exhausted by the other vehicles. Al the routes, which included this stop, is removed from the possible route list. This can result in stops not being visited, because there is no vehicle available, or so it seems. We can build up a route, which consist of the unrouted stops to insure a feasible solution, but this will result in a solution that is not the best.

- As mentioned previously, we cannot guarantee that Taillard's method will result in the best solution. If we add to that the additional complexity of our problem, it can really get time consuming to rebuild the solution from a set of feasible homogeneous vehicle routes. This implies that the heuristic method applied on the homogeneous vehicle solution will be applied a few times. With the available computer power as well as the complexity of the data sets we work with, it will be more effective to implement the vehicle selection method into the heuristic. Taillard found that for problem instances involving very few vehicles, there was a higher probability that a run would not produce a good or even a feasible solution.

2.3.5. Time Windows

The Vehicle Routing Problem with Time Windows (VRPTW) is by far the most popular implementation of the VRP. Our problem implements various extensions on the original idea of a time window. The customers to be visited can have multiple time windows. The vehicles to be used will also have available time windows that will allow the user to schedule certain vehicles for long hauls where

necessary. There exist a wide variety of implementation methods for the VRPTW.

A Hybrid Search Based On Genetic Algorithms And Tabu Search For Vehicle Routing

Ombuki et al, [39] presented a hybrid search technique based on meta-heuristics for approximately solving the VRPTW. The approach is two phased; a global customer clustering phase based on genetic algorithms (GAs) and a post-optimization local search technique based on Tabu search (TS). They also devised a new crossover operator for the VRPTW and compare its performance with two well-known crossover operators for VRPTW and related problems. Computational experiments show that the GA is effective in setting the number of vehicles to be used while the Tabu search is better suited for reducing the total number of distance travelled by the vehicles. Through their simulations, they conclude that the hybrid search technique is more suitable for the multi-objective optimization for the VRPTW than applying either the GA or Tabu search independently. We definitely take this from their research and will also implement a hybrid approach.

In this paper a hybrid search technique is proposed which is suitable for multi-objective optimization. Their approach is two phased; a global customer clustering phase based on genetic algorithm and a post-optimization local search technique based on Tabu search. The objective function states that costs should be minimized. In this case the objective is to minimize the number of vehicles used and the distance travelled to meet the demand of all the customers while not exceeding capacity of the vehicle and the latest time for serving each customer. Thus this problem can be treated as a multi-objective optimization problem.

In the GA, each chromosome in the population pool is transformed into a cluster of routes. The chromosomes are then subjected to an iterative evolutionary

process until a minimum possible number of clusters are attained or the termination condition is met. The transformation process is achieved by the routing scheme whereas the evolutionary part is carried out like in ordinary GAs, that is, in each generation, genetic operations, crossover and selection are applied upon chromosomes. We represent each chromosome as sequence of cluster of routes. A route is composed of a sequence of nodes (customers). Each chromosome represents a possible solution for the VRPTW.

The following figure shows the performance of the genetic algorithm compared to that of the Tabu search technique. In the case of Figure 3, the main objective under scrutiny is how GA and Tabu search performs respectively in defining the final number of vehicles to be used to service the customers for the VRPTW. Likewise, Figure 4 demonstrates their performance when the main objective observation is to minimize distance travelled. The vertical axis in both figures shows the number of customers not served. The more customers served, the better. From Figure 3 we observe that GA performs better than the Tabu search in searching the "optimal" number of vehicles to service the customers. As the figure shows, the GA manages to employ a smaller number of vehicles and also to serve more customers than the Tabu search approach.

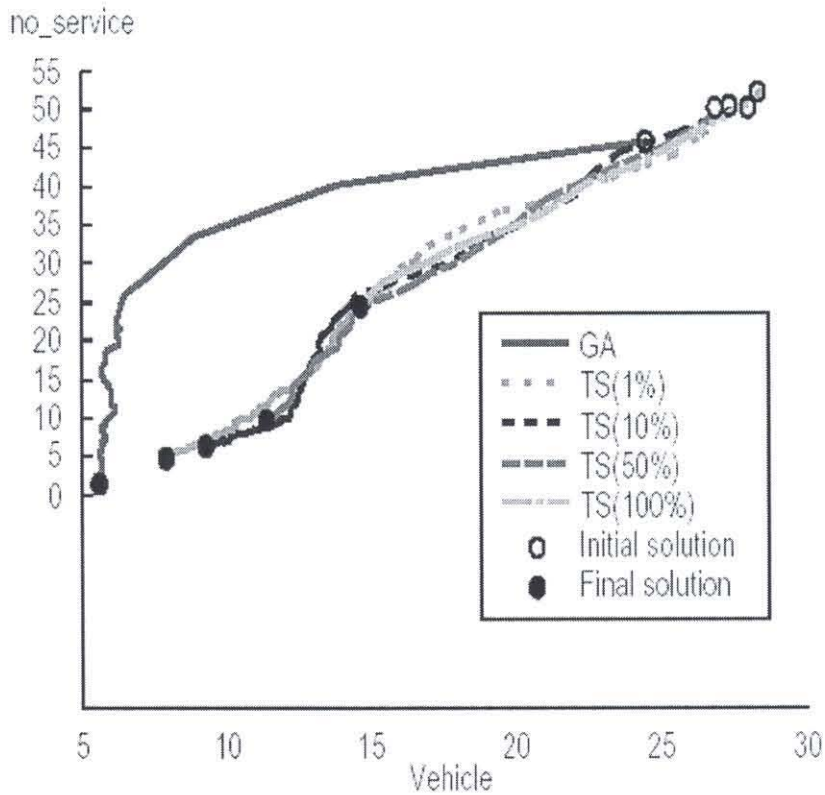

Figure 3: GA vs. Tabu Search for Minimizing Vehicles

On the other hand, Figure 4 depicts that the Tabu search outperforms the GA when it comes to minimizing the total distance travelled. Clearly, this is a case of conflicting objectives. In-order to reduce the travelled distance; one would need to increase the number of vehicles. On the other hand, to reduce the cost of employing more vehicles, one needs to increase the distance travelled per vehicle (which does not necessarily solve the problem as the cost of gas and other resources comes into play as well).

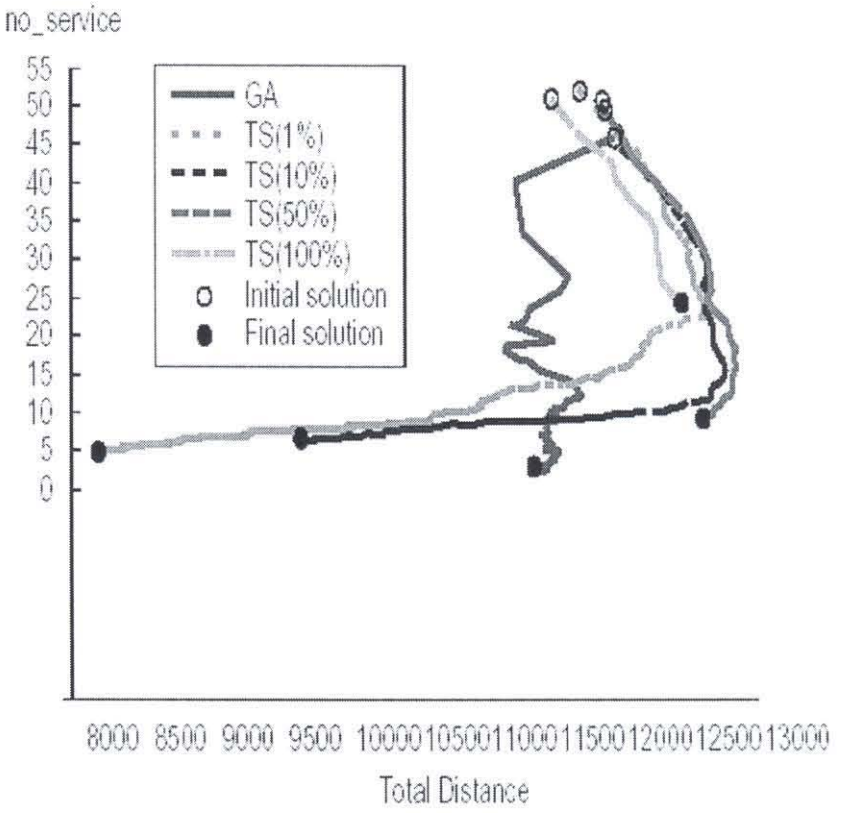

Figure 4: GA vs. Tabu Search for Minimizing Distance

In our problem we are not concerned about reducing vehicles as a main objective, although we would like to utilise a vehicle as good as possible. Instead of making use of GA for vehicle reduction, we implement methods to handle heterogeneous fleet, as well as multiple scheduling. The GA method in this implementation as a heuristic and not a meta-heuristic. What we are looking for is a method to handle the meta of our algorithm.

Although they do not specify the side constraints, except for the time windows, additional side constraints can be implemented and will affect the algorithm in testing for feasibility. We can expect similar results for our problem as in this instance.

A Network Flow-Based Tabu Search Heuristic for the VRP

Xu and Kelly [54] introduced a network flow model as a general local search strategy to solve the VRP. They used a straightforward model by relaxing the hard side constraints and introducing a dynamic penalty system, and efficiently update and frequently solve the network flow model to find the best customers to insert into new routes without the use of the generalized assignment problem. The penalty parameters are changed such that the feasibility of the search is controlled.

The network flow model implements Tabu Search restriction to prevent the method from getting trapped in local optima. TS restrictions with randomly generated tabu tenures are applied to them three neighbourhood moves: dropping a customer from its current route, inserting a customer into a different route and swapping two customers between routes. For the swap, in addition to Tabu restrictions on future swaps, the associated ejections and insertions are also subject to tabu restrictions. When a customer is moved to a new route, a tabu restriction that prevents its removal from that route is only activated when there are only a few customers (less than a pre-determined number) in the route.

From their implementation we conclude that each operation can have its own tabu tenure. Ideally we would like to set the tabu tenure during execution for each operation. We also conclude that the execution of an operation might result in tabu moves for other operations. We must identify the dependencies of operations beforehand.

Xu and Kelly [54] also implement an intensification strategy that we inherit from them based on advanced restart/recovery procedure. The set of best feasible solutions produced by the search are defined as elite solutions. A repository of elite solutions is maintained. Advanced restart is executed periodically during the late stages of the search. When restarting, the current solution is obtained from the repository and all tabu restrictions are released. This strategy is based on the assumption that there may exist short relinking paths in the search process from the restart points to new local or global optima. However, these paths may not be detected during prior search due to the tabu restrictions. The advanced restart/recovery strategy may find these paths and thereby lead the search to new local or global optima.

Vehicle Routing in Constraint Programming

De Backer and Furnon [18] consider constraint programming for solving VRPs. However, this raises many problems. Search in constraint programming is usually based on depth-first search. This means that the domains of each variable are monotonically reduced by propagation during the search. Although this approach can be useful for finding a first solution for the VRP, it is not practicable when an optimized solution is sought. This is the reason why much research has been devoted to the design and the implementation of local search techniques in the context of routing problems.

The paper presents basic principles for implementing local search techniques and meta-heuristics in constraint programming. These principles have been applied to Tabu Search. We consider the basic VRP with additional side constraints. Expressing such constraints as what we are considering, can be tedious, and yield problems with huge models, especially in the case of traditional linear programming (LP) models, or make programs solving VRP very complex and difficult to maintain.

In standard LP models, decision variables usually belong to a set of Boolean variables x_{ij}^{k} which take the value 1 if the vehicle k is used to travel from visit i to j. Therefore, these models use $O(mn_2)$ decision variables, where m is the number of vehicles and n is the number of visits to perform.

We implement the VRP with a number of variables that is linear (instead of quadratic) with respect to the number of visits. Each visit i is associated with two finite-domain variables $next_i$ and veh_i, representing respectively the possible visits following i and the vehicle serving visit i. This method allows us to quickly access the feasibility of a route by traversing only a part of the route depending on where the alteration took place.

De Backer and Furnon [18] devise a generic way of taking into account constraints on dimensions that can be a diverse as weight, time, or volume. They introduce the notion of a path constraint, which are similar to the way that we implement constraints on a route. Path constraints are able to propagate accumulated quantities such as time and weight along a vehicle tour.

We implement a similar method to test for feasibility of a route. Constraints are prioritised according to ease of calculation and importance on failing, e.g. to insert a node in a route, the vehicle capacity must be sufficient to accept the new node as well. It is quick to test the current capacity of the route plus the new load of the stop against the capacity of the vehicle.

Time Window Compatibility

Time Window Compatibility (TWC) refers to the compatibility of the time window(s) of one stop with regards to another. A good TWC figure indicates that the two nodes are likely to be inserted in sequence on the same route. In many cases two customers can be located next to each other, but their time

windows is not compatible. The trade-off between distance (i.e. cost) and time (i.e. customer delight) is an inherent part of the problem.

Insertion of stops in a heuristic fashion requires a selection process that result in a possible next stop. The TWC can assist us in ruling out infeasible stops from the start. We define the term neighbour for a stop. A neighbour is a stop that can be visited from the current stop. If we know that a stop is not a neighbour of the current stop, we do not even waste time of trying to implement that stop as a next stop. The neighbours of a stop are made up of all the time window compatible stops. We utilise the TWC principle as proposed by Van Schalkwyk [52], but we implement it in a different fashion. A discussion of the TWC follows and Chapter 3 will discuss the implementation of this concept in our solution.

The figure below illustrates a scenario where we evaluate the time adjacency of node *i* and node *j*. This scenario assumes that there will be a definite overlap in time windows between the two nodes. Other scenarios will subsequently be discussed.

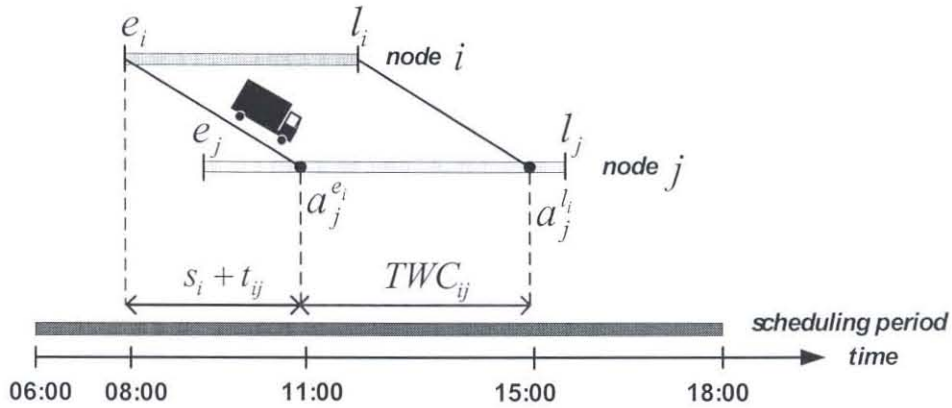

Figure 5: The basic TWC calculation - Scenario 0

Scenario 0: *IF* $a_j^{e_i} > e_j$ *AND* $a_j^{l_i} < l_j$

Customer i specified a time window (e_i, l_i) between 8:00 and 12:00, and customer j requires service between 9:00 and 16:00 (e_j, l_j). If serviced started at node i at e_i (the earliest feasible time), its arrival at j would be:

$$a_j^{e_i} = e_i + s_i + t_{ij}$$

In this scenario equals 11:00.

Similarly, al would be the arrival at j if service started at node i at the latest possible time (l_i):

$$a_j^{l_i} = l_i + s_i + t_{ij}$$

In this scenario equals 15:00.

The difference between $a_j^{e_i}$ and $a_j^{l_i}$ will yield the amount of time overlap between i and j:

$$TWC_{ij} = a_j^{l_i} - a_j^{e_i}$$

In this scenario it equals 4 hours. The significance of this value is that the bigger the overlap, the better we can insert the two nodes in sequence. This also ensures that the customer with a big overlap is routed first (more flexible).

A number of different scenarios will be illustrated in the following figures.

Scenario 1: If $a_j^{l_i} > l_j$

If the earliest arrival time at node j is inside the acceptable time window, but the latest arrival time is outside of the acceptable time window of node j, the two customers only partly overlap. The TWC_{ij} is then calculated by the following equation:

$$TWC_{ij} = l_j - a_j^{e_i}$$

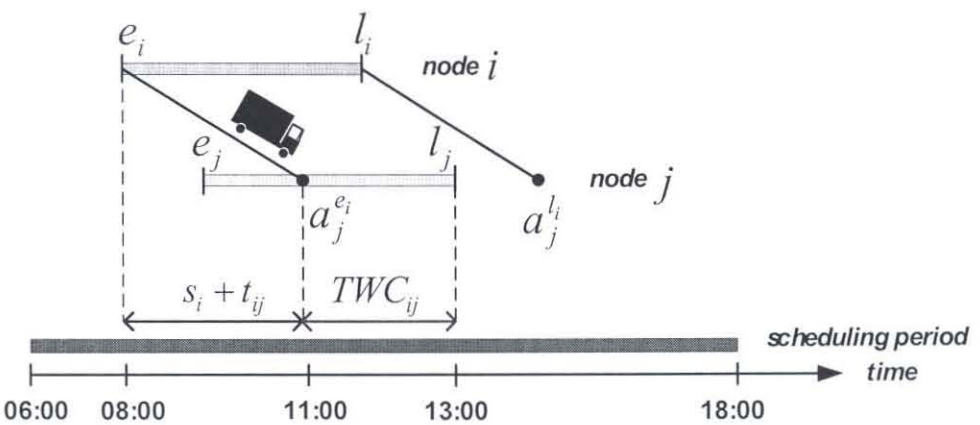

Figure 6: Scenario 1 TWC calculation

Scenario 2: If $a_j^{e_j} < e_j$

If the vehicle arrives at the earliest feasible time and this is before the acceptable time window of node j, and the arrival of the latest feasible time at node j is inside the acceptable time window, the two customers only partly overlap. The vehicle has to wait to service customer j. The TWC_{ij} is then calculated by the following equation:

$$TWC_{ij} = a_j^{l_j} - e_j$$

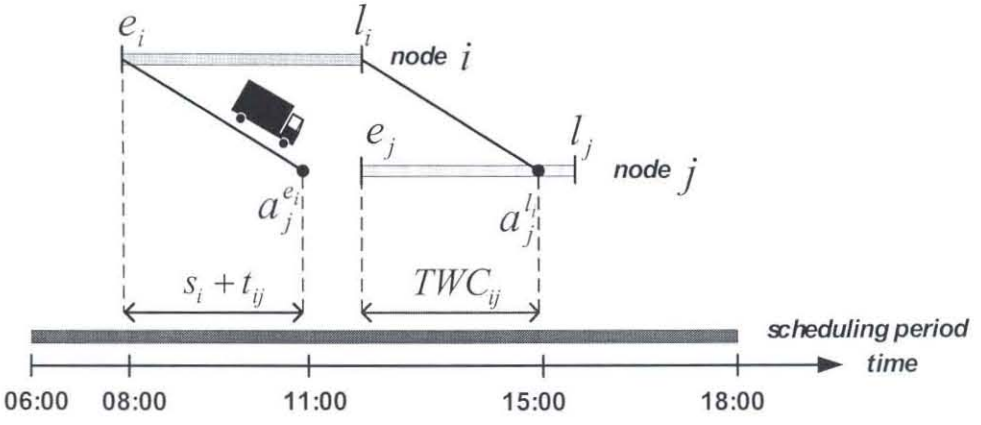

Figure 7: Scenario 2 TWC calculation

Scenario 3: If $a_j^{e_i} < e_j$ and $a_j^{l_i} < e_j$

If the latest arrival time at node j is earlier than the start of the acceptable time window at node j, the vehicle always waits at node j, irrespectable of the arrival time at node i. The arrival at j is always before its acceptable start time. This value will be negative, and calculated as follows:

$$TWC_{ij} = a_j^{e_i} - e_j$$

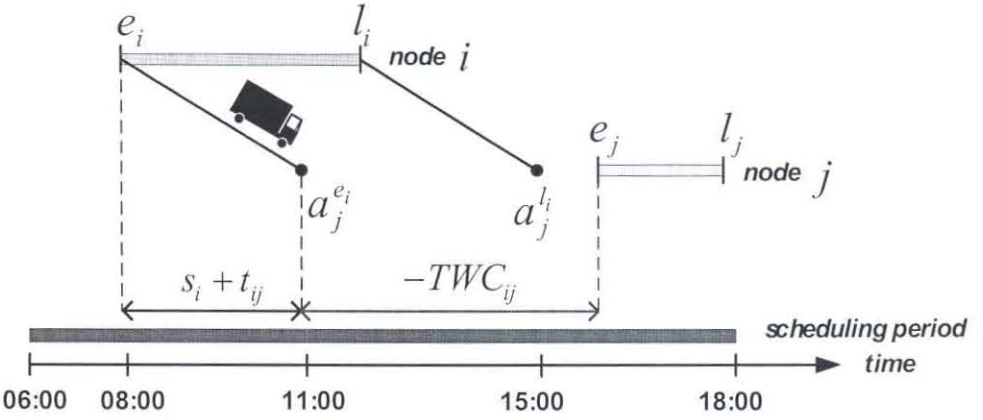

Figure 8: Scenario 3 TWC calculation

Scenario 4: If $a_e > l_j$ and $a_l > l_j$

If the arrival time at j is always bigger than the latest acceptable time at j, the node-combination is infeasible. The nodes forming part of this combination will typically be eliminated before starting the algorithm, as they can obviously not be included in the current route under construction.

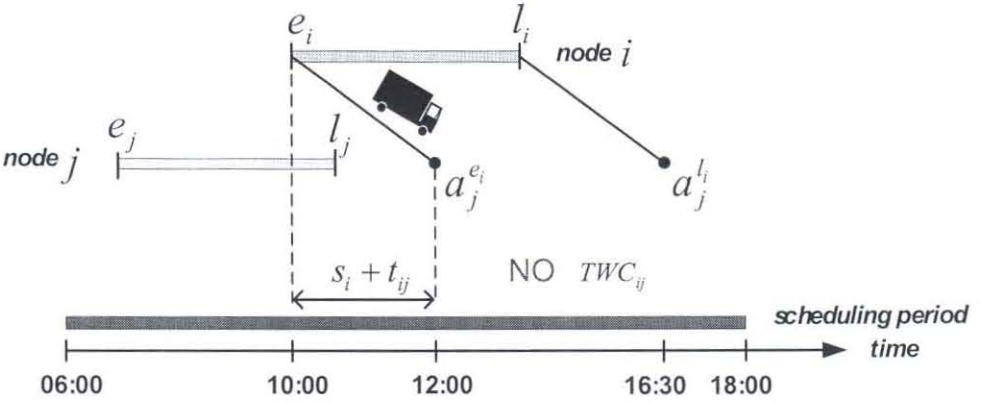

Figure 9: Scenario 4 - infeasible combination

Chapter 3

3 PROBLEM SOLVING METHODOLOGY

Solving the vehicle routing problem in its basic format is already an NP-hard problem. Exact methods have proved to be inefficient and time-consuming in trying to solve this problem. Previous attempts on solving the VRP have indicated that heuristic methods result in the best feasible solution in an acceptable time. When we add additional constraints to the basic VRP, we increase the difficulty of the solution exponentially. We must also consider the size of the data set their needs to be optimised.

Heuristic methods search only part of the solution space. This result in the quicker termination of the algorithm, but does not guarantee a best solution. Previous results have shown that heuristic methods can achieve optimal or near optimal results repeatedly. The meta-heuristic method has a guidance procedure of some sort to help it traversing through the solution space. The guidance procedure is dependent on the type of heuristic selected for the solution, as well as additional knowledge from the problems space implemented by the algorithm. This additional information about the problem beforehand can assist the algorithm in more effective search paths. A meta-heuristic is the implementation of a heuristic method with a guidance procedure.

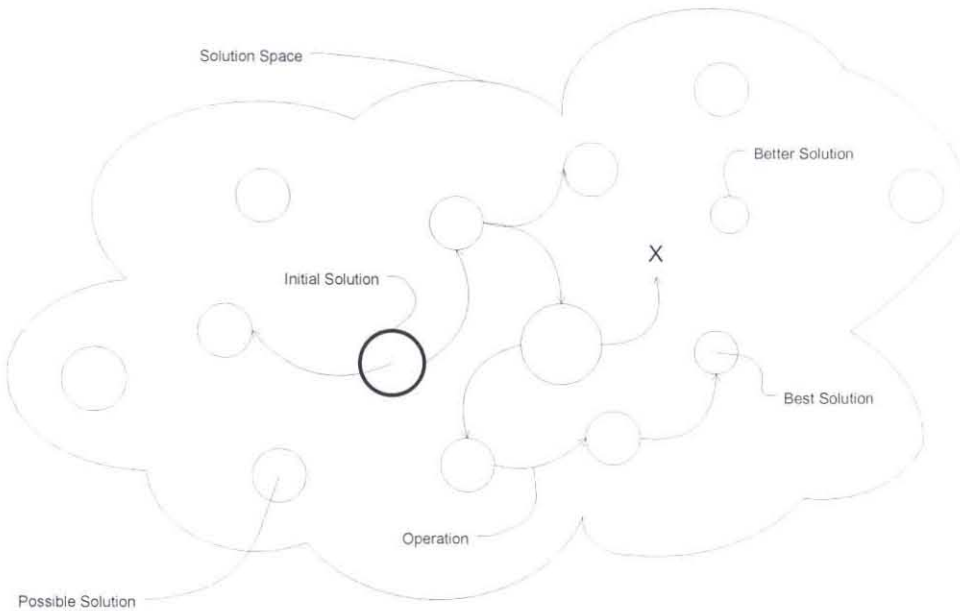

Figure 10: Solution Space

Figure 10 explains the methodology of heuristic methods for solving the particular problem. The solution space consists of all possible solutions for the specific problem. Theoretically we can develop an algorithm that has the ability to generate all of the possible solutions such as branch and bound methods. As we have already seen, this method will take an eternity on the complex problem that we are trying to solve. A meta-heuristic can search effectively through the solution space.

A circle which size reflects the total cost of the solution represents a solution. The smaller the circle, the better the solution. This indicates that there are possible solutions that is not cost-effective and which we do not want to consider as an end result.

Let S be a set of solutions to a particular problem, and let f be a cost function that measures the quality of each solution in S. The neighbourhood N(s) of a

solution s in S is defined as the set of solutions which can be obtained from s by performing simple modifications. Roughly speaking, a local search algorithm starts off with an initial solution in S and then continually tries to find better solutions by searching neighbourhoods. A local search process can be viewed as a walk in a directed graph G=(S,A) where the vertex set S is the set of solutions and there is an arc (s,s') in A if and only if s' is in N(s). By considering the cost function as an altitude, one gets a topology on G=(S,A).

The efficiency of a local search method depends mostly on the modelling. A fine-tuning of parameters will never balance a bad definition of the solution set, of the neighbourhood, or of the cost function.

The topology induced by the cost function on G=(S,A) should not be too flat. The cost function can be considered as an altitude, and it therefore induces a topology on G=(S,A) with mountains, valleys and plateaus. It is difficult for a local search to escape from large plateaus since any solution that is not in the boarder of such a plateau has the same cost value as its neighbours, and it is therefore impossible to guide the search towards an optimal solution. A common way to avoid this kind of topology on G=(S,A) is to add a component to the cost function which discriminates between solutions having the same value according to the original cost function.

Our evolutionary metaheuristic makes use of the well-known two-stage and multi-start local search (MLS) frameworks. In two-stage framework the initial solution created in the first stage is subsequently improved in the second one.

In the first stage we generate an initial solution with the help of a construction heuristic, in this case we make use of the sequential insertion heuristic (SIH). This method results in a solution that is feasible but not necessarily the best. The

feasibility of the solution ensures that it existing our solution space (see the initial solution in Figure 10).

The improvement stage traverse from our current position to a neighbour's solution. Because solutions do not truly exist in our environment, we need to generate a new feasible solution. This is done by applying an operation on the current solution. As we progress it can happen without an already existing solution is generated by an operation. This can result in cycles in our search path, which leads to revisiting existing solutions and result in unnecessary computational time. One of our objectives will be to prevent such cycling. After a specified number of iterations we have visited a number of solutions from which the best solution is kept. The solution is not necessarily the best solution for the problem, but represents the best-visited solution. Our goal is to guide the search path in such a way that we cover as wide as possible area of the solution space.

From the figure we can see that the path to the best solution might have to go through a not so good solution before the best solution is reached. Operations applied on a solution can result in a not feasible solution. We can consider this as a stepping-stone towards the next solution, or it can be seen as a waste of computational time.

The improvement phase is implemented with the Tabu Search Method. Tabu search has a rationale that is transparent and natural: its goal is to emulate intelligent uses of memory, particularly for exploiting structure. Since we are creatures of memory ourselves, who use a variety of memory functions to help thread our way through a maze of problem-solving considerations, it would seem reasonable to try to endow our solution methods with similar capabilities.

The following sections will discuss in more detail the specific methods used to traverse through the solution space. It will also point out where knowledge about

the problem beforehand can have an effect on the implementation of the solution. The sections consist of the problem representation in objects, the approach of the solution, a discussion on the construction heuristic and improvement heuristic.

3.1. Objects.

In the previous chapter we presented the problem in a mathematical model. This model has the purpose of describing the parameters of the problem as well as the conditions it has to meet. Implementing a solution for the problem is not as easy as describing it. This section will explain the components we utilise for solving the problem. The solution was designed in an object orientated way.

The object model is divided into two areas. Model will describe the problem objects or the input data. The second model will describe the alterations on the problem objects and the additional objects required to produce a solution. An object consists of properties, methods and relations.

3.1.1. Problem objects.

This section will discuss the mapping from the input data to the objects in the solution. We need to identify all the objects represented in the input data. Let us consider the vehicle routing problem again.

The basic VRP consist mainly of a depot, stops and vehicles. A depot can be seen as a specific stop with certain properties.

Stops

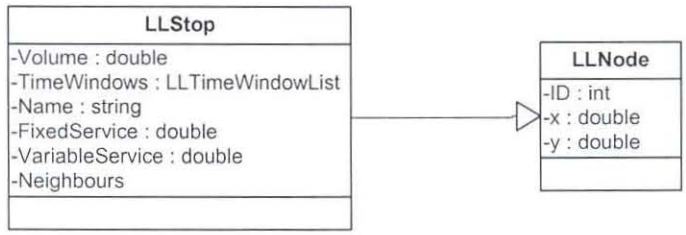

Figure 11: Problem object Stop

The above object represents a stop. A stop must comply with the basic functionality of a graph node. The figure indicates that a stop inherits all the properties and methods of a node. The properties of a stop is as follows:

- ID - a unique value to identify the stop.
- X, Y - the spatial representation of the node
- Volume - the volume that a stop will utilise on a vehicle
- Time Windows – a list of available time windows that a stop can be visited in.
- Name – a descriptive name for the stop for display and report purposes
- Fixed Service – the fixed service time for a stop in minutes. This represents the stopping time required at a stop without loading or unloading anything.
- Variable Service – this represents the volume per minute rate of loading or unloading goods at the stop. The total service time at the stop consist of the fixed service time + (volume * variable service time)

- Neighbours – this is a list of neighbours that a vehicle can visit from a stop. In the basic VRP this list will consist of all the other stops. In our problem that includes time windows, it might happen that it will never be feasible for a vehicle to travel from one stop to another because of time window compatibility (see description of time window compatibility), which basically means that the following stop has time windows that ends before the current stop's time windows begin. Science has shown that we cannot travel back in time and thus we will not consider this stop as a neighbour.

Operations required by the problem model for stops can be defined as followed:

- Travel Time - working with the restriction of time widows, we need to know that time it will take to travel from one stops to another to ensure that we arrive at a feasible time. We implement travel time between stops in a matrix. One of the additional constraints to our problem is the requirement to calculate the travel time depending on the time of the day. The travel time function accepts the two stops in the travelling sequence and the time of departure from the first stop. See this section on the cost matrix for further detail.

- Distance - distance is calculated in a similar way as travel time. Distance is also dependent on time of day because the travel time between two stops determines the route between the stops. This means basically that a quicker route might not be the shortest.

Depot

The properties and methods of a depot is exactly the same as for a stop. In defining a depot, we define a single stop. Travel time and distance calculations applied on the depot in the same manner as for a stop.

Our solution considers only one depot, which implies that all the existing vehicles and stops belong to that depot. Extending this problem to a multi- depot problem would require the depot object to be reconstructed by adding a stop list as well as vehicle list to the depot object.

Vehicle

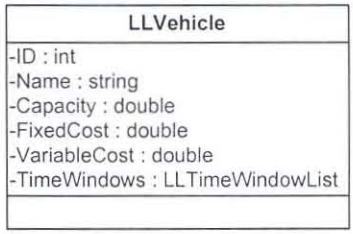

Figure 12: Problem object Vehicle

The vehicle object in our implementation consist of the following properties:

- ID - a unique key for identifying the vehicle
- Name - a descriptive name for display and reporting purposes
- Capacity - the total volume that a vehicle is capable to handle
- Fixed Cost - the cost of utilising this vehicle without even travelling
- Variable Cost - the running cost of the vehicle. Part of the cost of the route is calculated by Fixed Cost + (Variable Cost * Distance).

- Time Windows - a list of available time widows that the vehicle can be utilised.

There does not exist specific operations for a vehicle in the problem object model.

Time Windows

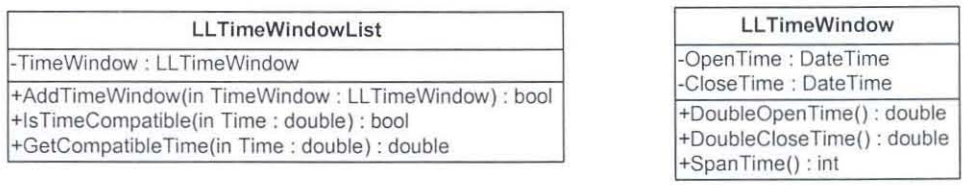

Figure 13: Problem object Time Window

Time widows play an important role in the problem. All of the problem objects, namely depot, stops and vehicles, are associated with a time window list to indicate availability for the object's specific function.

Time window consist basically of an open and close time. This time is saved in a datetime format to allow for implementing problems that span across multiple days. Operations on the time window includes:

- DoubleOpenTime - returns the number of minutes after specific date time from a fixed time. This is done to allow the algorithm to work in a linear reference environment. Let us for example say that the open time is 07:00 on today's date. Calculating the linear time consist of the difference between the open time and today at midnight, which results in 7 hours. Converting the hours to minute's results in a linear open time of 7 * 60 = 420. If the open time was specified as yesterday at 07:00 the difference between today at midnight and the open time is –17 hours. Converting

the hours to minute's results in a linear open time of -17 * 60 = -1020. Although the value is negative is still valid for a linear scale.

- DoubleCloseTime - returns the number of minutes after a specific date time, same as DoubleOpenTime.

- SpanTime - returns the difference between the open and close time in minutes.

What we can see from the time window properties is that our linear timescale consists of minutes. The fixed point on the scale to calculate the linear values from is today's date.

The time window list object consists of a list of time windows. Operations on this list include:

- IsTimeCompatible - this function accepts a time and determines if there exists a time window that include the time, i.e. the time is after the open time and before the close time for a specific time window in the list.

- GetCompatibleTime – this function accepts a time and calculates the earliest available time according to the time window list. If no such time exists, an exception is thrown, which indicates incompatible time.

3.1.2. Solution Objects

This section will give an overview of the solution objects used in the algorithm. It is important to understand this basic building blocks in order to see how the algorithm functions. Solution objects consist of extensions of problem objects to handle new information required by the solution, as well as help objects that play a role in solving the problem.

Route Vehicle

The implemented solution focus on deterministic data, i.e. all the demands and vehicles are available and known before the start of the solution. In terms of the vehicles the algorithm will not propose a best-suited fleet from a set of vehicles, but accept the vehicles as existing and ready to use according to their specifications. It can be simplified by allocating a route to a vehicle before we even start. The solution is therefore made up of a set of vehicles that contain routes.

One of the additional requirements of the problem is to allow for multiple routes on a vehicle. A vehicle can thus have multiple routes.

A vehicle with routes will be the main output of the system. A route vehicle is the input vehicle with routes associated to it.

Routes

A route can be seen as a sequence of stops that is visited by a particular vehicle at a specific time.

1.1.2.3 Route Stops

The determining of a best solution relies mainly on the handling of the stops. | A route stop consist of a stop with additional info such as:

- Arrival Time – the time a vehicle arrives at a stop
- Wait Time – the time a vehicle must wait at a stop before it can start servicing the stop.
- Service Time – as specified by the stop service time.

- Departure Time – the time the vehicle leave a stop for its next stop. This must be equals to the Arrival Time + Wait Time + Service Time.

- Next Stop – An indication on where to go next in the route. This method is the principle method of providing information on the route. Adding or deleting a stop from a route is made easy by just replacing the next stop. Adding a new stop requires replacing the current stop's next stop with the new stop and the new stop's next stop to the current stop's next stop. Deleting is as easy as setting the previous stop's next stop value to the current stop's next stop value. This only indicates the method of inserting and deleting a stop from a route and not the validity of the move.

VRP Base

The main purpose is to solve the VRP. There exist several ways to solving a VRP. This object is the base object for the solution. The object contains all the necessary data and manipulates all the necessary methods applied on the data. The end result of the algorithm is the VRP object, which contains multiple solutions.

Cost

Cost is defined as the cost in terms of distance and travel time from one stop to another. A cost matrix is used for storing the values.

The solution implements a cost function with time windows to represent the difference of cost on a link depending on the time. This basically result in a cost function that is a function of the time of day. When the algorithm requests a travel time from the cost function, the function first determines the cost matrix to use. This is done by finding a cost matrix, which time windows will contain the time provided. The cost for that time is returned.

It is important to notice the influence of such time dependent cost function in the solution. The advantage is that a more accurate route can now be constructed, which is very important for the success of the algorithm. When a vehicle travels from point A to point B, it will definitely take him longer during traffic peak periods. The use of an average travel time on a link will no be sufficient to take care of this problem. When a vehicle travels during peak time, his actual arrival time at the customer will be later than planned. Although the vehicle might make up this time during the off-peak time, the use of multiple time windows can result in a lateness that fall between two time windows, which result in additional wait time, which makes it more difficult to make up during the off-peak times.

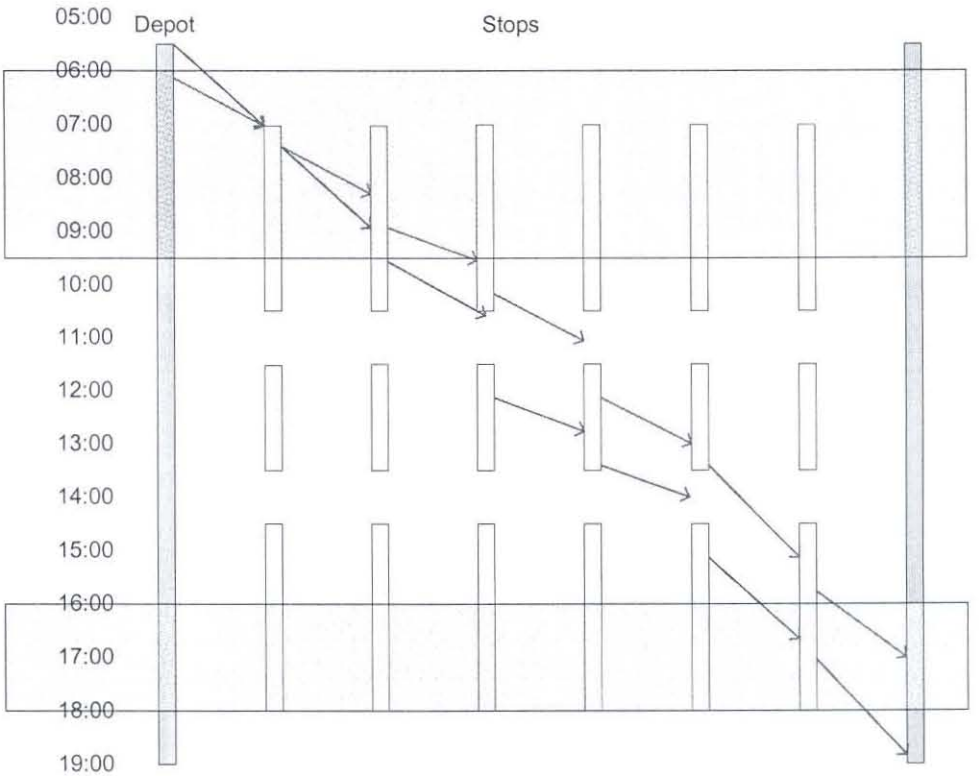

Figure 14: Peak and Off-Peak travel time influence

Figure 14 explains the importance of a time dependent cost function in the solving of the VRP. The figure represents a typically delivery day with stops that has similar time windows. The patterned areas represent peak traffic time. A route is constructed from the depot on the left back to the depot on the right.

The green arrow line represents the route making use of an average travel time on a link. The red line represents the actual travel time. Starting of, we can immediately see that the average route departs later that the actual route. This is because the departure time from the depot is determined by the open time of the first stop. The slope of the red line is steeper than the green one, which indicates a longer time to travel from the depot to the first stop in the actual route.

The algorithm will ensure that the arrival time at the first stop is as early as possible. In the above case, both routes arrive at the open time of the first stop. The service time is not affected by the cost function and both routes depart from the first stop at the same time.

During the peak travel time, the actual route requires a bit more time to travel than the average travel time. At stop 3, the actual time arrives too late to be serviced in the first time window and has to wait for the second time window to take effect. Although the actual travel time is quicker than the average time during off-peak periods, the aggregated loss due to lateness cannot be recovered. This is mainly due to the synchronisation of the stop time windows.

The example above is proof that we need to implement a time dependent cost function in the algorithm to produce more realistic results.

The VRP is a NP-hard problem, which suggest that it is difficult to solve. Heuristic methods can provide feasible solutions in reasonable time, but additional constraints will increase computational time. The addition of a time

dependent cost function requires the algorithm to recalculate the travel time between two stops every time a new stop is added to the route or a stop is removed from the route. This is necessary for all stops after the added or removed stop, as the addition of a stop will alter the arrival time of all subsequent stops.

Solution

A solution object represents a possible solution to the VRP problem. The solution contains route vehicles and their corresponding routes and stops, as well as an orphan list of stops. A solution object is used to generate more solutions from through an operation.

Although the algorithm considers all the main influential parameters, we cannot ignore the human factor. There might still exist a preference from the user regarding a specific solution. During the execution of the algorithm the proposed methodology requires a list of solutions to be able to traverse through the solutions space. We propose that the algorithm does not only present the user with the best to solution found, but provide the option of selecting one of the best solutions. Practical implementation has shown that the best calculated solution might not always be the most feasible for the client. This might be because of the customer driver relationships, driver knowledge of areas, etc.

Construction Heuristic

The proposed solution requires some possible solution to start working from. There exist multiple methods of constructing an initial solution. In a later section the selected construction heuristic namely the Sequential Insertion Heuristic (SIH) will be discussed. The algorithm can function from an existing solution. In those cases, the construction heuristic would not be necessary.

Working in the ASP environment implies dynamic acquisition of data from clients. The solution has to take into consideration the possible extension of the current implementation, i.e. there might exist a better construction heuristic for the specified problem. For that specific reason we propose the implementation of a construction heuristic in the main algorithm. This will allow the addition of other construction heuristics in the future. The current construction heuristic already produces multiple solutions for the improvement heuristic to work on.

Improvement Heuristic

The implementation of an improvement heuristic is the focus area of this research topic. The VRP object contains an Improvement Heuristic method. As in the case of the construction heuristic, the VRP is force the existence of such a method, but does not determine the implementation detail.

3.1.3. Problem Helper Methods

This section will discuss the systematic approach in solving the problem. Although the focus of this thesis is on designing a new VRP solution, we cannot ignore the implementation environment. The ASP environment has a major influence on the line and implementation of the solution algorithm. The main reason is because of the unpredictability of the data.

The next paragraphs will discuss information flow and manipulation through the process.

Input Data and Object Generation

The first step towards a feasible solution is to acquire data from the client. There exist multiple methods of transferring data from the client information service to the ASP server. This is the topic of another study.

What is important is that the data must be complete. This means that the incoming data must contain all the necessary information. In addition, we must know where the incoming data is headed for, e.g. the client must specify which value from a stop is the demand and which is the time windows etc.

The client data must now be constructed in the defined objects. The algorithm requires data that is relevant to one depot and one instance of a routing schedule. This means that a stop will only be visited once during the time windows specified.

After this step, the algorithm will contain all the necessary data.

Solution methods

As explained in previous sections, a route consists of a sequence of stops. The manner in which the structure is maintained is important in the manipulation procedures of the algorithm. This paragraph describes basic actions allowed on a solution. The implementation of the construction and improvement heuristics will depend on the stability of these actions.

Route stop addition

As mentioned previously in the discussion of the time dependent cost function, the addition of a stop on a route has several consequences on the subsequent stops.

The addition of a stop in a route results in this shift of the arrival time of subsequent stops, which can result in time window incompatibility, i.e. the arrival time is not sufficient anymore to be able to serve the stop in its available time windows. An action of inserting a stop in a route that result in incompatible time windows must flag the route as invalid.

The removal of a stop on a route has less dramatic results, i.e. if a route was valid before the removal of a stop, it can still be valid. It might not be as efficient, but it will still exist in the solution space.

The addition of a stop on a route also has an effect on the vehicle volume. Adding a stop increase the volume required on the vehicle. The addition of a stop can result in a route that exceeds the vehicle capacity. This action must flag the route as invalid.

The removal of a stop result in the decrease in the required volume for the vehicle. The removal of a stop from a route cannot result in a vehicle that exceeds capacity.

It is important to know that the weight and arrival time calculations have to be executed on each insertion and removal of a stop in a route. The implementation of these methods must be effective.

Vehicle stop addition.

The addition of a stop on a route has an effect on the overall routes associated with the vehicle.

When a stop is added on a route, the route's departure and arrival time from the depot change. This can result in a delay in the departure of a next route from the depot. The new departure time for the next route can result in incompatible time windows at stops, or even an incompatible time window for the route vehicle. The addition of a stop on a route can result in the invalidity of subsequent routes and the route must be flagged accordingly.

Time Window Compatibility

The concept of a time window compatibility matrix as proposed by van Schalkwyk, [52] has not been proven, but has a logic sense to it. The calculation

of such a matrix can be done at the beginning of the algorithm, which adds to the setup time, but not the running time.

An aspect not catered for in the proposal of the TWCM is the variation in the travel time depending on the time of the day. The addition of variable travel time adds some complexity to the problem. In Figure 15 we show effect of the variable travel time.

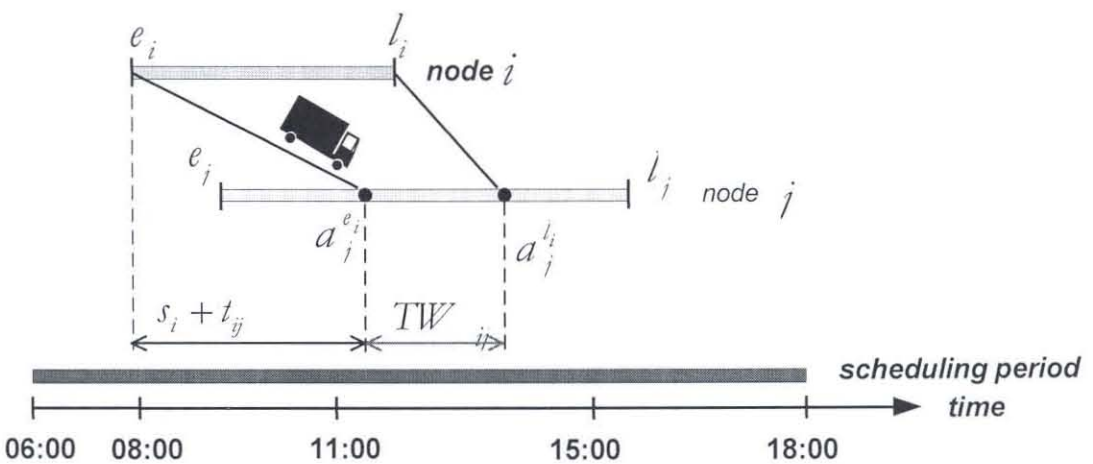

Figure 15: Variable Travel Time on Time Window Compatibility

From the figure we can depict the effect of the variable travel time. In this implementation, the travel time function is not a continuous function, but a disjunctive function consisting of constant times at specific intervals. In our calculation of the TWC, we need to overlay the travel time function's time windows with that of the source stop. We determine travel time from the source stop's departure time.

3.2. Approach

The approach consists of different phases, which will be discussed in more detail in the following paragraphs. The first phase consist of the generation of the required distance and time matrices for specific time periods. The second phase is the generation of an initial solution through a construction heuristic. This is necessary for the improvement heuristic that follows. The improvement heuristic will follow the guidelines of the Tabu Search. The heuristic will search for a good solution by diversifying and intensifying the solution area. After a predetermined number of iterations, or if a termination parameter is met, the post optimization phase will ensure that the current best solution is optimised to its local minimum.

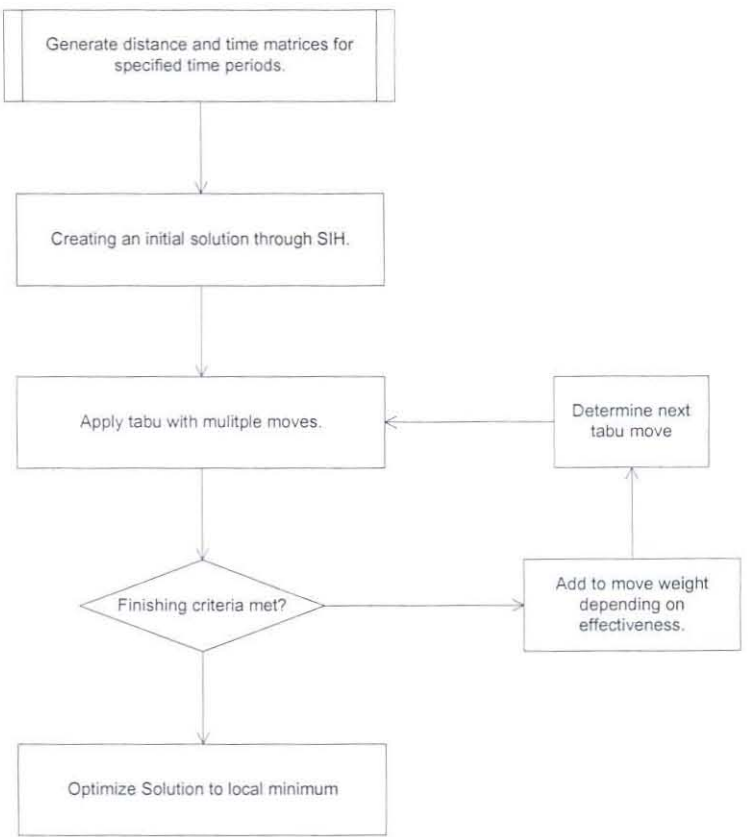

Figure 16: Algorithm Phases

3.3. Initial Solution

High quality initial heuristics often allow local searches and metaheuristics to achieve better solutions more quickly. Marius Solomon was one of the first researchers to consider the VRPTW. He designed and analysed a number of algorithms to find initial feasible solutions for the VRPTW (Solomon, 1987). His sequential insertion heuristic (SIH) gave very good results in most environments, and most current heuristic methods make use of this heuristic (or a variation thereof) to effectively find a feasible starting solution.

Each customer i has a known demand q_i to be serviced (either for pickup or delivery) at time b_i chosen by the carrier. Because time windows are hard, b_i is chosen within a time window, starting at the earliest time e_i and ending at the latest time l_i that customer i permits the start of service. A vehicle arriving too early and customer j, has to wait until e_j. If t_{ij} represents the direct travel time from customers i to customer j, and s_i the service time add customer i, then the moment at which service begins at customer j, b_j, equals $\max\{e_j, b_i + s_i + t_{ij}\}$ and the waiting time w_i is equal to $\max\{0, e_j - (b_i + s_i + t_{ij})\}$.

After initialising the route, the insertion criterion $c_1(i, u, j)$ determines the cheapest insertion place for all remaining, unrouted customers between two adjacent customers i and j in the current partial route $(i_0, i_1, \ldots, i_m)$. Each route is assumed to start and end at the depot $i_0 = i_m$. The indices $p = 1, \ldots, m$ are used to denote a customer's position in the route. The insertion cost is a weighted average of the additional distance and time needed to insert the customer in the route. The parameters α_1, α_2, μ and λ are used to guide the heuristic.

Inserting customer u between i and j increases the length of the route by the distance insertion, $d_{iu} + d_{uj} - m\,d_{ij}$. After inserting a customer u between the

adjacent customers i and j, a push forward can be calculated for each consecutive node k,

$$PF_k = b_k^{new} - b_k$$

in which b_k (b_k^{new}) denotes the beginning of service at customer k in the route before (after) inserting customer u. The value of PF_k is maximal for the direct successor $k = j$ of u. The sequential insertion heuristic uses the maximal push forward to measure the time needed to insert customer u in the route, the so called time insertion.

The next step of the sequential insertion heuristic decides on which customer to insert the route. The selection criterion c_2 (i, u, j) selects the customer for which the cost difference between insertion in the current or a new route is the largest. This customer is inserted in its cheapest insertion position in the current route. If all remaining unrouted customers have no feasible insertion positions, a new route is initialised and identified as the current route.

We extend the Solomon criteria by utilising the neighbour stop information in testing for a suitable stop to add to the route. Using only stops that have a time window compatibility value, reduce the number insertion positions to test for each stop. When testing for the insertion position in the current route fails because of the TWC, inserting customer u between adjacent nodes for the rest of the route will fail as well. This method will increase the speed of the construction heuristic without diminish the quality of the result.

We also extend the criteria by a Push Backward if a customer is inserted between the depot and the first customer as proposed by Dullaert and Bräysy (2003) [21]. If customer u is inserted between the depot $i_0 = i$ and the first customer $i_1 = j$, a push backward is introduced in the schedule. Since all vehicles are assumed to

leave the depot at the earliest possible time e_0, and travelling from i to j takes t_{ij} units of time, a waiting time of $\max\{0, e_j - t_{ij}\}$ is generated at $j = i_1$. Unlike the waiting time at all other customers $i_r, p < r \leq m$ in the route, it is fictitious. After finishing the route, it can be eliminated by adjusting the depot departure time. High waiting times stored at customers that used to be scheduled at the first position during the solution construction, cannot be removed this easily. By assuming all vehicles leave the depot at e_0 and by equalling the time insertion to the maximum push forward, the time needed to insert a customer before $i_1 = j$ can be underestimated. It may even be wrongly equalled to zero.

We also extend the Push Backward to incorporate the vehicle time windows. Inserting a customer u as the first stop in the route advances the departure time at the depot depending on the open time of the depot, the best available time of the vehicle and the open time of the customer u. The vehicle would leave the depot at $\max\{b_i=0, b_k, b_j - t_{ij}\}$ where $b_i=0$ is the open time of the depot, b_k the open time of the vehicle and $b_j - t_{ij}$ the open time of u retracting the travel time from i to j.

3.4. Improvement Heuristic

Chapter 2 discussed heuristic techniques we considered for implementing a solution for the specified VRP problem. It suggested the use of a meta-heuristic technique. Meta-heuristics use information of the problem environment and the nature of the objective function to direct the search process to regions that promise better solutions.

Although there exist many alternatives in selecting the appropriate tool, the success of these methods depends on many factors, like their ease of implementation, their ability to consider specific constraints that arise in practical applications and the high quality of solutions they produced.

A distinguishing feature of Tabu search is its exploitation of adaptive forms of memory, which equips it to penetrate complexities that often confound alternative approaches. The rich potential of adaptive memory strategies is only beginning to be tapped, and the discoveries that lie ahead promise to be as important and exciting as those made to date. Principles that have emerged from the TS framework give a foundation to create practical systems whose capabilities markedly exceed those available earlier. Conspicuous features of Tabu search are its dynamic growth and evolving character, which are benefiting from important contributions by many researchers.

Tabu search provides a range of strategic options, involving various levels of short term and long-term memory. Consequently, it can be implemented in corresponding levels ranging from the simpler to the more advanced. Generally, the more advanced versions exhibit the greatest problem solving power, though simple ones often afford good results as well. The convenience of building additional levels in a modular design, allowing a TS procedure to be evolved from the "ground up," is a feature that also provides a way to see and understand the relevant contributions of different memory based strategies.

Implementing a specific strategy for the specified problem is complicated by the fact we cannot or should not rely on the manner of the problem. As mentioned in the introduction, input data can vary from long haul to short haul, long time windows or short multiple time windows, heterogeneous fleet of similar fleet. To solve the VRP with all its side constraints and unpredictable in put data, we implement new operations and add some statistical selection method in the guidance algorithm.

3.4.1. Operations

Insert Operator

The insert operator tries to insert an orphan stop into an existing route. The method loops through the orphan list of the current solution and calculates a best insertion position. The orphan stop's neighbours are tested for insertion cost. This is done by selecting a neighbour, determining the route the neighbour belongs to and calculates the cost of inserting the orphan stop after the neighbour. If the neighbour is an orphan itself, the test is not done. The method locates a set of closest geographic neighbours from the stop and test the validity of the insertion of the orphan stop after the neighbour stop. The move is accepted if the insertion is valid.

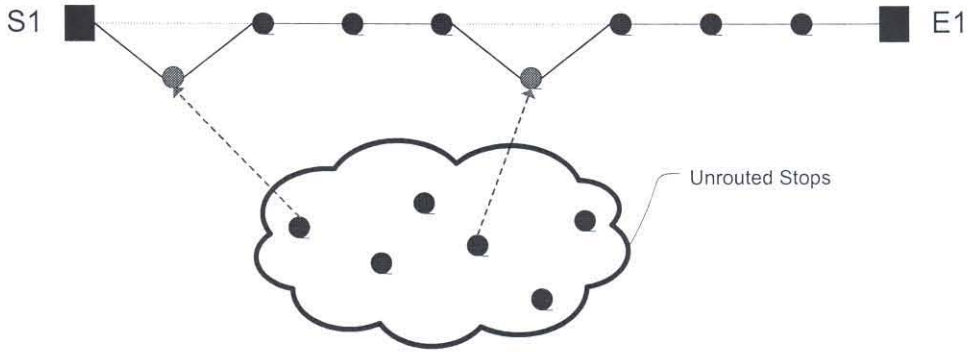

Figure 17: Insert Operation

Tour depletion operator

The purpose of this move is to reduce the number of vehicles required to serve all the stops. If it is possible to remove a vehicle, the probability that total distance will decrease is high. It might not be the result in some situations, but the heuristic also depends on diversification.

The procedure looks for the vehicle that contains the least number of stops allocated to routes for the vehicle and is not Tabu. We qualify the routes of a vehicle for removal if the number of stops is less than a percentage of the average number of stops in all the vehicle routes. This is done on the assumption that stops and vehicles have similar characteristics. The difference between stops in terms of volume is assumed to be in a reasonable tolerance.

The first step is to select a tour for depletion according to the criteria specified.

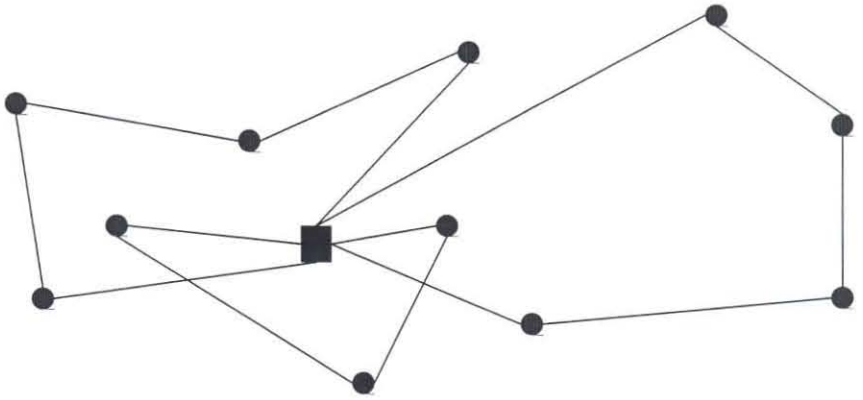

Figure 18: Tour Depletion Step 1

The tour is removed from the solution and the stops belonging to the tour is added to the orphan list.

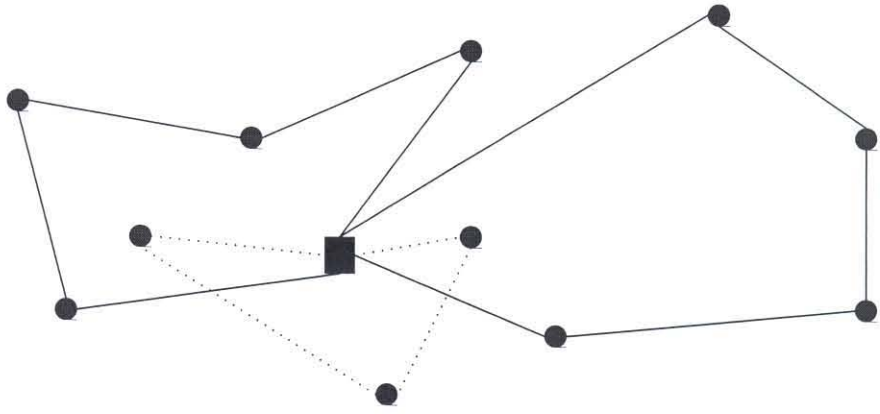

Figure 19: Tour Depletion Step 2

The insert operator is executed to insert the newly created orphans into existing routes.

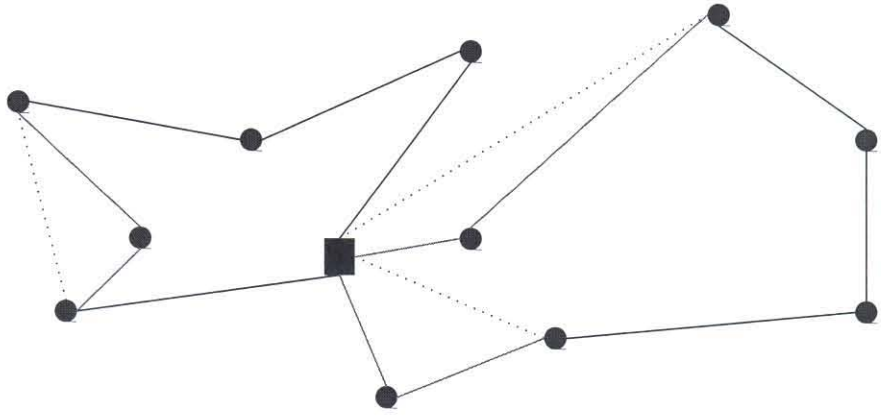

Figure 20: Tour Depletion Step 3

An additional criteria for the tour depletion operator to execute is the non-existence of orphans in the solution. We implement the logic before we even start with actions on the operator, as we assume that if an orphan exists, the current solution is already in such a state that the current route vehicles cannot service all

the stops. The meta-heuristic guidance algorithm must execute other operations to optimise the solution that tour depletion is possible.

Relocate operator

The relocate operator (Or-opt) removes one stop from a route and inserts it into another route. The implementation group routes to a vehicle and therefore we randomly select a vehicle to add a stop to. Next we randomly select one of the vehicle routes. For each stop on the current vehicle route, an attempt is made to insert a neighbour of the current stop on the current vehicle route. The neighbour is relocated from its route to the current route.

The relocate operator can relocate a stop from the same route to another position.

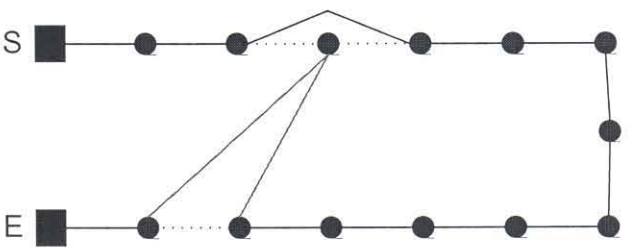

Figure 21: Relocate on same route

Or relocate a stop from one route to another.

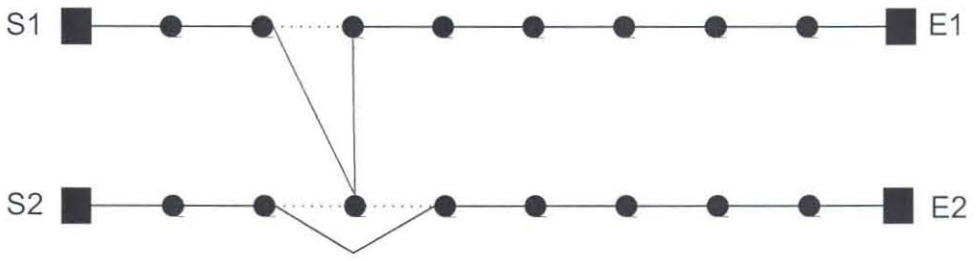

Figure 22: Relocate between routes

Exchange Operator

The exchange operator randomly selects a vehicle and corresponding route. The neighbours of the selected route's stops are tested for exchange between the corresponding routes. The operator acts on single stops from different or same routes only.

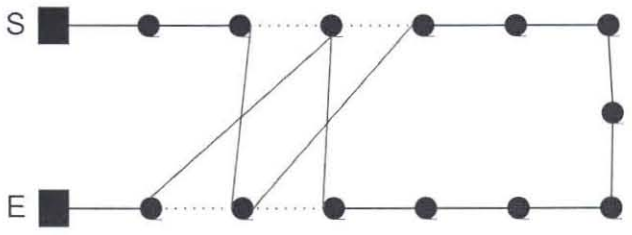

Figure 23: Exchange on single route

The exchange from one route to another simulates a relocate from the one route to the other and vice versa.

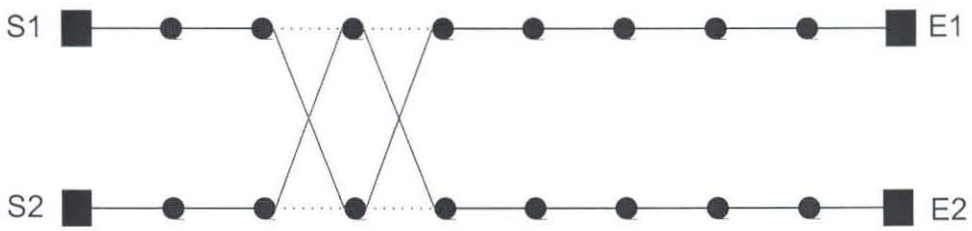

Figure 24: Exchange between routes

Cross operator

This operator cuts two routes at a position and swaps the second part of the routes. This is done by selecting a source vehicle and a source route randomly. Each stop in the source route is tested for the move. The stop's neighbours are tested for validity by checking if the stop is not on the same route. If not, the source route consisting of the stops up to the selected stop is combined with the target route consisting of the stops from the neighbour stop to the end to form a new route. The second new route consist of the target route from the beginning to the stop before the neighbour stop and the source route from the stops after the selected stop to the end. If the swap is valid in the current Tabu environment, it will be accepted.

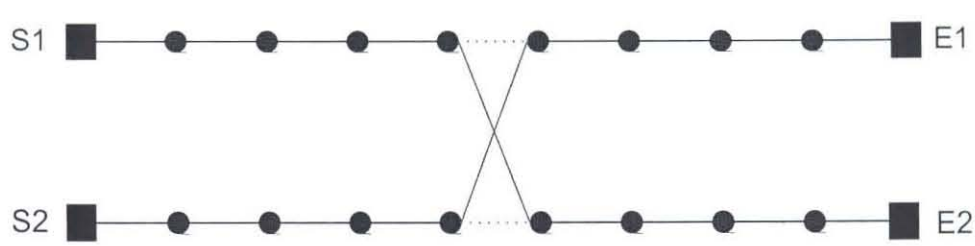

Figure 25: Cross operation

Vehicle Fit

This operator exchange vehicles on routes. The operation is added to handle the heterogeneous fleet optimization problem. A vehicle can be swapped between routes if the capacity and time windows allow for the routes qualify.

If there exist vehicles that have not been used, the vehicles can be tested on existing routes to result in better optimization. Tour depletion can result in a more effective vehicle to become available, and the vehicle fit operator will reinsert an available vehicle in the solution.

Double Fit

The operation tries to fit routes or segments of routes as additional routes on a vehicle. This action will result in the use of fewer vehicles.

The double fit operator has the purpose of filling up a vehicle to its time window capacity. The operator will test form time available on a vehicle and if there exist a continuous time that is greater than a minimum time specified, the operator can look for stops that fit in that time frame. If a route can be constructed to fill the open time slot, the move is accepted and results in other routes that have fewer stops. This move can now result in probable tour depletion after some optimization on the routes.

3.4.2. Guidance Algorithm

Meta-heuristics use information of the problem environment and the nature of the objective function to direct the search process to regions that promise better solutions. The implementation of the guidance algorithm has an important effect on the effectiveness of the algorithm.

The implementation of the guidance algorithm utilise aspects from different sources. A simulated annealing approach is followed in an oscillating fashion.

Neighbourhood search methods are also selected randomly in a statistical learning fashion. Each operation has its own tabulist.

Statistical Selection

The implementation of all the specified operations can lead to inefficient computational time utilisation. Depending on the manner if the input data, some operations can be more effective than other, or can be ineffective in situations.

When the input data has customers with tight time windows, the capacity of the vehicle does not really play an important role in the solution, as the vehicle does not have time to visit enough stops to load the vehicle to capacity. The double fit operation will not be effective on these types of data. The statistical selection will eliminate the use of this operation.

The idea of the statistical selection is to determine the success rate of an operation. When we randomly select an operation, the probability of the selection of a specific operation depends on the success rate. When we start the improvement heuristic, we assign an equal value to the success rate of all the operations in the list. On the first iteration, the probability for an operation to be selected is the same for all. If the operation completes successful, we increase the success rate by a value depending on the type of success. This increase will not have a major effect in the beginning, but after a number of iterations, the more successful operation's success rate will increase, and that will increase the probability of the selection.

Simulated Annealing

Another control mechanism implemented by the guidance algorithm is derived from the simulated annealing procedure. In the modified version of SA, the algorithm starts with a relatively good solution resulting from a construction heuristic. Initial temperature is set at $T_S = 100$, and is slowly decreased by

$$T_k = (T_{k-1})/(1+\tau\sqrt{T_{k-1}}) \quad (1)$$

Where T_k is the current temperature at iteration k and t 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 the scheme, gives fairly good results in much less time. The algorithm attempts solutions in the neighbourhood of the current solution randomly or systematically and calculates the probability of moving to those solutions according to:

$$P(accepting\ a\ move) = e^{(-\Delta/T_k)} \quad (2)$$

This is a modified version of the annealing equation, where Δ = C'(S) - C(S), C(S) is the cost of the current solution and C'(S) is the cost of the new solution. If $\Delta < 0$ the move is always warranted. One can see that as the temperature cools, the probability of accepting a non-cost-saving move is getting exponentially smaller. When the temperature has gone to the final temperature T = 0.001 or there is no more feasible moves in the neighbourhood, we reset the temperature to

$$T_r = \max(T_s/2, T_b) \quad (3)$$

where T_r is the reset temperature, and was originally set to T_s, and T_b is the temperature at which the best current solution was found. Final temperature is not set at zero because as temperature decreases to infinitesimally close to zero, there is virtually zero probability of accepting a non-improving move. Thus a final temperature not equal but close to zero is more realistic. The Tabu Search is used to search the local neighbourhood.

3.5. Conclusions

This chapter describe the design of a solution algorithm that is capable to solve the VRP in an ASP environment. The additional constraints imposed by the ASP environment are incorporated in the design of the algorithm.

The problem is partially solved by the introduction of new operations on the solution as well as extensions of current existing operations. The guidance algorithm implements multiple operations, which allows it to be effective on all types of input data. The statistical selection of operations is believed to improve the effectiveness of the algorithm.

Chapter 4

4 COMPUTATIONAL RESULTS

The VRP problem is NP-hard and making use of heuristic methods results in unpredictable results. Heuristic methods are non deterministic which contribute to the complexity in measuring the effectiveness of the method applied on the problem.

The VRP with additional side constraints is a complex problem that complicated basic rules specified for the guidance algorithm of the applied meta-heuristic. Depending on the distribution of data points, time windows, peak and off-peak travel times, vehicle capacity and demand per stop, the algorithm must adapt to the data environment during the execution to result in an acceptable feasible solution. To achieve this, we implemented a multiple operation selection method. We projected that there must be an effective operation in our list of operations on the data environment. In the previous chapter, we discussed the methods and proof theoretically that the proposed solution will be effective. In this chapter we will discuss the impact of the operations on the problem, as well as the additional advantage obtained by using these operations in combinations.

The implementation of the algorithm consist of two phases: the initial solution make use of the Sequential Insertion Heuristic to construct a set of initial routes and the improvement heuristic consist of a hybrid method based mainly on the Tabu Search technique and the Simulated Annealing method. Although we are interested in the improvement heuristic, we will present the results of the construction heuristic to indicate the efficiency of the improvement heuristic.

Results will be presented for two types of problems:

1. The traditional Solomon benchmark problems will be solved to indicate the efficiency of the algorithm with known results.

2. A real-life problem will be solved and efficiency will be discussed.

The chapter will discuss the results of the initial solution, the effect of the individual operations of the improvement heuristic and the results of the improvement phase.

4.1. Solomon's Benchmark Problems

Solomon generated six sets of problems. Their design highlights several factors that affect the behaviour of routing and scheduling algorithms. They are: geographical data; the number of customers serviced by a vehicle; percent of time-constrained customers; and tightness and positioning of the time windows.

The geographical data are randomly generated in problem sets R1 and R2, clustered in problem sets C1 and C2, and a mix of random and clustered structures in problem sets by RC1 and RC2. Problem sets R1, C1 and RC1 have a short scheduling horizon and allow only a few customers per route (approximately 5 to 10). In contrast, the sets R2, C2 and RC2 have a long scheduling horizon permitting many customers (more than 30) to be serviced by the same vehicle.

The customer coordinates are identical for all problems within one type (i.e., R, C and RC). The problems differ with respect to the width of the time windows. Some have very tight time windows, while others have time windows, which are hardly constraining. In terms of time window density, that is, the percentage of

customers with time windows, he created problems with 25, 50, 75 and 100 % time windows.

The larger problems are 100 customer euclidean problems where travel times equal the corresponding distances. For each such problem, smaller problems have been created by considering only the first 25 or 50 customers. We only consider the larger problems.

4.1.1. Initial solution.

High quality initial heuristics often allow local searches and metaheuristics to achieve better solutions more quickly. We implemented the sequential insertion heuristic (SIH) proposed by Marius Solomon. We extended the Solomon criteria by utilising the neighbours stop information in testing for a suitable stop to add to the route. We also extended the criteria by a push backward if a customer is inserted between the depot and the first customer as proposed by Dullaert and Bräysy (2003).

When we start a route, the selection of the first node can be done according to the following criteria:

- Selecting the node that has the latest departure time.
- Selecting the node that has the earliest arrival time.
- Selecting the node that is the furthest from the depot.
- Selecting the node that is the closes to the depot.

The seed node criteria results in a different solution set according to the seed selection. The selection of a seed vehicle can also result in a different solution and we select the vehicle according to the following criteria:

- The vehicle with the smallest capacity.
- The vehicle with the least running cost.

Combining these two criteria, we result in eight possible initial solution generation methods. Although the implementation of all eight methods contributes to additional computation time, we can motivate the decision by the following:

- The input data is unpredictable and we cannot beforehand decide which method will be the best for the input data.
- The better the initial solution, the quicker the improvement phase. The time spend on the additional seed criteria will be made up in the improvement phase.
- The use of a neighbour list and the greedy nature of the sequential insertion heuristic result in a fix time for the initial solution.

The following table shows the initial results for the 56 Solomon benchmark problems according to the seed criteria. Because Solomon uses homogeneous fleet, only the stop criteria are considered. The highlighted text shows the best result achieved.

Problem Class C								
Problem	Latest Departure		Earliest Arrival		Furthest		Closest	
C101	10	923.70	10	880.47	10	880.47	10	928.22
C102	11	1193.46	10	997.74	11	1151.06	10	1075.08
C103	11	1317.31	11	1536.23	12	1501.95	10	1081.56
C104	10	1135.85	10	1419.31	11	1098.36	10	1059.59
C105	10	878.78	10	934.36	10	934.36	10	932.38
C106	11	1073.75	10	1068.90	10	1076.65	10	968.58
C107	10	928.74	10	1066.52	10	1066.52	10	1017.70
C108	10	871.57	10	1122.68	10	1114.62	10	1121.52
C109	10	910.28	10	1152.90	10	1285.48	11	1188.59
Problem	Latest Departure		Earliest Arrival		Furthest		Closest	
C201	3	895.38	3	1023.26	3	826.15	3	838.65
C202	3	1180.34	3	1727.11	3	1778.32	3	1774.13
C203	3	1173.25	3	1572.67	3	2091.21	3	1965.51
C204	3	1235.70	3	1498.34	3	1524.62	3	1509.96
C205	3	789.79	3	1318.07	3	1026.28	3	1170.94
C206	3	934.87	3	1456.09	3	1413.57	3	1349.79
C207	3	884.44	3	1040.77	3	1082.25	3	1140.17
C208	3	815.97	3	1201.14	3	1108.66	3	1205.91

Table 2: Solomon Initial Solution Results Class C

Problem Class R								
Problem	Latest Departure		Earliest Arrival		Furthest		Closest	
R101	20	1857.93	23	2303.99	25	2293.20	24	2301.59
R102	19	1792.59	20	2095.62	20	1913.25	21	1956.94
R103	15	1553.58	17	1777.33	19	1777.82	17	1694.45
R104	12	1283.22	13	1516.91	12	1334.73	12	1358.23
R105	15	1534.40	16	1804.79	16	1802.51	16	1883.03
R106	15	1457.51	17	1776.22	16	1714.83	15	1715.28
R107	13	1336.79	14	1591.55	13	1488.27	14	1549.74
R108	10	1174.06	11	1284.84	12	1385.20	11	1237.10
R109	14	1423.01	14	1645.27	15	1641.27	14	1696.22
R110	12	1332.66	14	1620.57	15	1682.08	13	1577.98
R111	13	1344.17	15	1672.66	15	1652.95	13	1606.67
R112	11	1167.79	12	1475.42	12	1436.07	11	1335.74
Problem	Latest Departure		Earliest Arrival		Furthest		Closest	
R201	4	1791.78	5	1633.98	5	1822.54	5	2043.84
R202	4	1603.75	5	1703.38	5	1623.01	5	1570.04
R203	4	1325.15	4	1505.26	4	1602.95	4	1518.06
R204	3	1054.39	3	1146.17	3	1183.05	3	1107.48
R205	4	1551.95	4	1461.61	4	1467.55	4	1533.07
R206	3	1358.68	3	1364.04	4	1501.83	3	1378.20
R207	3	1205.44	3	1213.78	3	1272.68	3	1279.97
R208	3	954.38	3	985.00	3	908.49	3	945.51
R209	4	1441.55	4	1409.81	4	1339.33	4	1260.75
R210	4	1384.77	4	1548.22	4	1510.95	4	1478.19
R211	3	1080.89	3	1200.61	3	1173.58	3	1213.90

Table 3: Solomon Initial Solution Results Class R

Problem Class RC								
Problem	Latest Departure		Earliest Arrival		Furthest		Closest	
RC101	16	1929.02	17	2186.36	18	2065.91	16	2050.29
RC102	15	1789.29	16	2134.40	17	1900.65	17	2235.96
RC103	13	1613.99	14	1924.30	15	1765.69	15	1960.02
RC104	12	1363.74	13	1731.69	13	1524.04	13	1677.67
RC105	16	1805.33	18	2299.15	19	2236.09	17	2146.35
RC106	14	1581.39	15	1940.90	14	1932.27	16	2135.60
RC107	13	1607.96	14	1881.29	15	1896.47	14	1823.81
RC108	12	1340.10	13	1728.38	13	1626.48	13	1639.01
Problem	Latest Departure		Earliest Arrival		Furthest		Closest	
RC201	5	2213.00	5	2273.59	5	2272.32	5	2131.14
RC202	5	1943.42	5	2203.85	5	1953.77	5	2031.00
RC203	4	1727.98	4	1595.85	4	1692.00	4	1758.04
RC204	3	1217.82	4	1449.52	4	1464.28	3	1184.48
RC205	6	1940.44	5	2137.55	5	2396.53	5	2151.34
RC206	4	1691.69	4	1723.34	4	1631.19	4	1595.74
RC207	4	1731.50	4	1690.61	4	1491.13	4	1627.06
RC208	3	1275.21	3	1347.44	3	1347.62	3	1564.00

Table 4: Solomon Initial Solution Results Class RC

4.1.2. Improvement Phase

The previous paragraph has shown the effectiveness of the individual operators. The purpose of the improvement phase is to combine these individual operators such that we can achieve effective improvements. The utilisation of the operators in random combination with each other result in a robust method that achieve results faster.

The following table shows the results compared to the best-published Solomon results as well as the initial result the improvement heuristic started from.

Problem Class C								
Problem	Initial Solution		Improvement			Best Published		
C101	10	880.47	10	828.94	5.9%	10	828.94	0.0%
C102	10	997.74	10	871.32	12.7%	10	828.94	5.1%
C103	10	1081.56	10	916.83	15.2%	10	828.06	10.7%
C104	10	1059.59	10	911.85	13.9%	10	824.78	10.6%
C105	10	878.78	10	827.55	5.8%	10	828.94	-0.2%
C106	10	968.58	10	840.19	13.3%	10	828.94	1.4%
C107	10	928.74	10	827.55	10.9%	10	828.94	-0.2%
C108	10	871.57	10	827.55	5.1%	10	828.94	-0.2%
C109	10	910.28	10	829.74	8.8%	10	828.94	0.1%
Problem	Initial Solution		Improvement			Best Published		
C201	3	826.15	3	588.88	28.7%	3	591.56	-0.5%
C202	3	1180.34	3	623.46	47.2%	3	591.56	5.4%
C203	3	1173.25	3	625.46	46.7%	3	591.17	5.8%
C204	3	1235.70	3	685.10	44.6%	3	590.6	16.0%
C205	3	789.79	3	617.45	21.8%	3	588.88	4.9%
C206	3	934.87	3	629.63	32.7%	3	588.49	7.0%
C207	3	884.44	3	587.89	33.5%	3	588.29	-0.1%
C208	3	815.97	3	592.93	27.3%	3	588.32	0.8%

Table 5: Class C Solomon Solution[4]

[4] Source: Solomon M. [45]

Problem Class R								
Problem	Initial Solution		Improvement			Best Published		
R101	20	1857.93	20	1670.13	10.1%	19	1645.79	1.5%
R102	19	1792.59	19	1576.81	12.0%	17	1486.12	6.1%
R103	15	1553.58	15	1316.31	15.3%	13	1292.68	1.8%
R104	12	1283.22	11	1061.90	17.2%	9	1007.24	5.4%
R105	15	1534.40	15	1455.08	5.2%	14	1377.11	5.7%
R106	15	1457.51	14	1292.28	11.3%	12	1251.98	3.2%
R107	13	1336.79	12	1174.00	12.2%	10	1104.66	6.3%
R108	10	1174.06	9	1030.87	12.2%	9	960.88	7.3%
R109	14	1423.01	13	1284.32	9.7%	11	1194.73	7.5%
R110	12	1332.66	13	1205.48	9.5%	10	1118.59	7.8%
R111	13	1344.17	13	1239.26	7.8%	10	1096.72	13.0%
R112	11	1167.79	11	1059.78	9.2%	9	982.14	7.9%
Problem	Initial Solution		Improvement			Best Published		
R201	5	1633.98	4	1335.55	18.3%	4	1252.37	6.6%
R202	5	1570.04	4	1200.26	23.6%	3	1191.7	0.7%
R203	4	1325.15	3	972.59	26.6%	3	939.54	3.5%
R204	3	1054.39	3	842.54	20.1%	2	825.52	2.1%
R205	4	1461.61	3	1133.02	22.5%	3	994.42	13.9%
R206	3	1358.68	3	985.94	27.4%	3	906.14	8.8%
R207	3	1205.44	3	948.50	21.3%	2	893.33	6.2%
R208	3	908.49	2	845.94	6.9%	2	726.75	16.4%
R209	4	1260.75	4	930.43	26.2%	3	909.16	2.3%
R210	4	1384.77	3	1019.45	26.4%	3	939.34	8.5%
R211	3	1080.89	3	862.42	20.2%	2	892.71	-3.4%

Table 6: Class R Solomon Solution

Problem Class RC								
Problem	Initial Solution		Improvement			Best Published		
RC101	16	1929.02	16	1742.62	9.7%	14	1696.94	2.7%
RC102	15	1789.29	15	1625.30	9.2%	12	1554.75	4.5%
RC103	13	1613.99	13	1403.99	13.0%	11	1261.67	11.3%
RC104	12	1363.74	12	1212.92	11.1%	10	1135.48	6.8%
RC105	16	1805.33	16	1706.53	5.5%	13	1629.44	4.7%
RC106	14	1581.39	14	1502.00	5.0%	11	1424.73	5.4%
RC107	13	1607.96	12	1318.22	18.0%	11	1230.48	7.1%
RC108	12	1340.10	12	1240.27	7.4%	10	1139.82	8.8%
Problem	Initial Solution		Improvement			Best Published		
RC201	5	2131.14	4	1474.86	30.8%	4	1406.91	4.8%
RC202	5	1943.42	4	1298.28	33.2%	3	1367.09	-5.0%
RC203	4	1595.85	3	1081.34	32.2%	3	1049.62	3.0%
RC204	3	1184.48	3	883.53	25.4%	3	798.41	10.7%
RC205	6	1940.44	5	1311.93	32.4%	4	1297.19	1.1%
RC206	4	1595.74	4	1162.03	27.2%	3	1146.32	1.4%
RC207	4	1491.13	4	1106.24	25.8%	3	1061.14	4.2%
RC208	3	1275.21	3	920.17	27.8%	3	828.14	11.1%

Table 7: Class RC Solomon Solution

Figure 26 displays the results in graphical format. The results are within reasonable margin from the best-published results. We must take into account that the best-published methods were achieved by various methods, i.e. for a specific problem instance, a specifically designed algorithm were applied on the problem. The comparison confirms the ability of our algorithm to perform reasonable across different problem instances.

In some instances our algorithm improved on the best-published result. From problem RC202 we can see a 5% improvement on the best published. It must be

noted that the cost function was set only on distance for these instances, which could resulted in higher total cost. We can see that from the difference in number of vehicles in problems R211 and RC202.

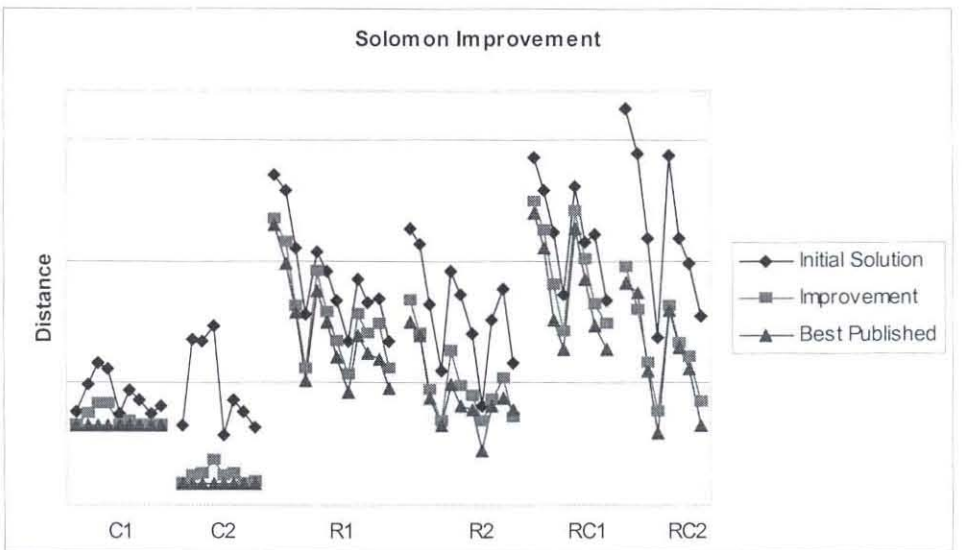

Figure 26: Solomon Improvement Comparison

4.2. Operation Results

Our algorithm was designed for the specific purpose of implementing it in the ASP environment. This environment is unpredictable in terms of input data, as well as cost factors. The idea of controlling specific operations through a meta heuristic had to be supported by a set of effective operations. Driven by the Tabu methodology, we were looking for operations that can assist as in both intensification and diversification. For this purpose we utilised some of the existing operations and designed new operations for the specific environment.

To ensure integrity of the system we tested each operation on its own to ensure that the operation acts according to expectation as well as resulting in useful neighbourhood solutions.

4.2.1. Insert Operator

This operation was added to ensure that we have viable routes by adding all the orphans available on the existing routes, or by creating new routes if the first is not viable. The insert operator has no definite improvement result, but works in combination with the tour depletion operator.

4.2.2. Tour depletion Operator

This operation was added to ensure diversification and optimisation by removing a vehicle from the current solution. This will force the application to optimise without the specific vehicle if possible, else creating a new route.

4.2.3. *Relocate Operator*

This operation is mostly affected on optimising a current solution. Depending on the deterioration tolerance, it will move a stop from one route to another. The following graph shows that this operation does not have a high feasibility rate, even though the deterioration tolerance for the specific situation was not set. This means that any viable solution was acceptable to the problem, even if it results in a worse solution than the current best. What we can see from the graph is the ability after this operation to optimise.

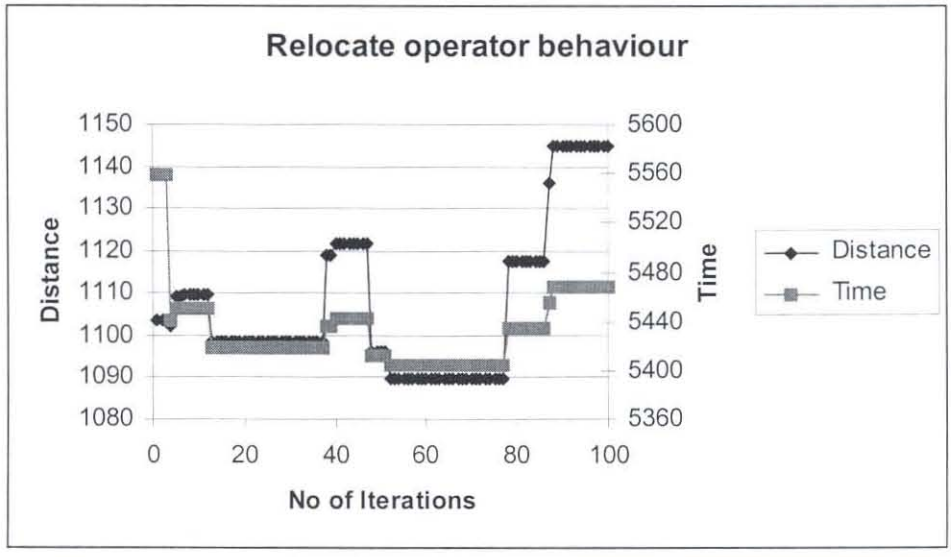

Figure 27: Relocate operator behaviour

4.2.4. *Exchange Operator*

The purpose of this operation is to swap two stops from different routes or within the same route with each other. The action can result in a better time utilisation or distance of the route. As can be seen from the graph below, this operation yields a feasible solution regularly. We can also see that the difference

in the time or distance from the previous solution is not as big as with the relocate operator. The graph indicates that this operation is important to finding the local minimum.

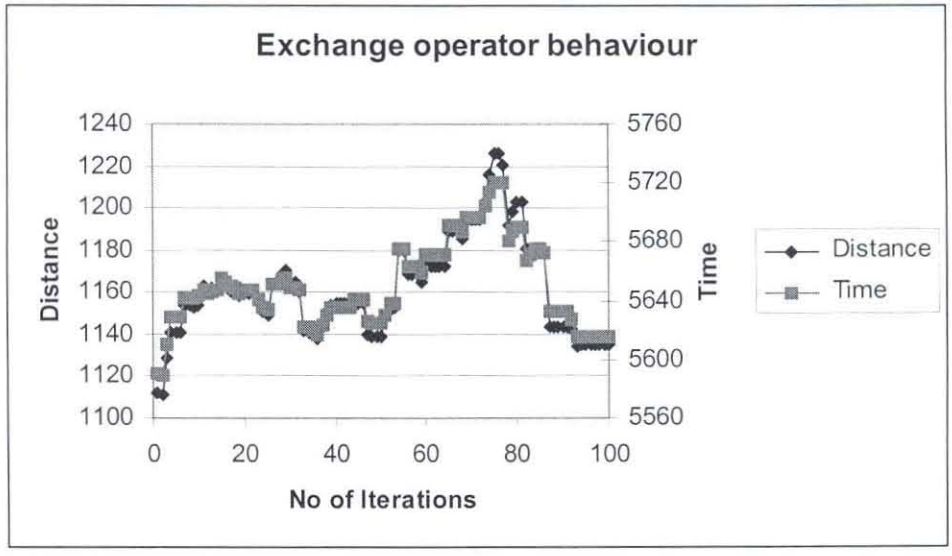

Figure 28: Exchange operator behaviour

4.2.5. 2-Operator

This operation takes two routes and cut them at specific positions and joins part one of route one with part two of route two and part two of route one with part one of route two. The implementation selects a target stop on route one and search for at a feasible swap route by traversing through its neighbours. As we can see from the graph, the move result in bigger changes from the previous solution, but has only a limited set of the viable moves. This can be seen in the latter part of the graph where the distance and time stays constant for long periods of iterations. We conclude that this is a result of the Tabu list that does not allow for previous moves to be repeated and no new moves exist.

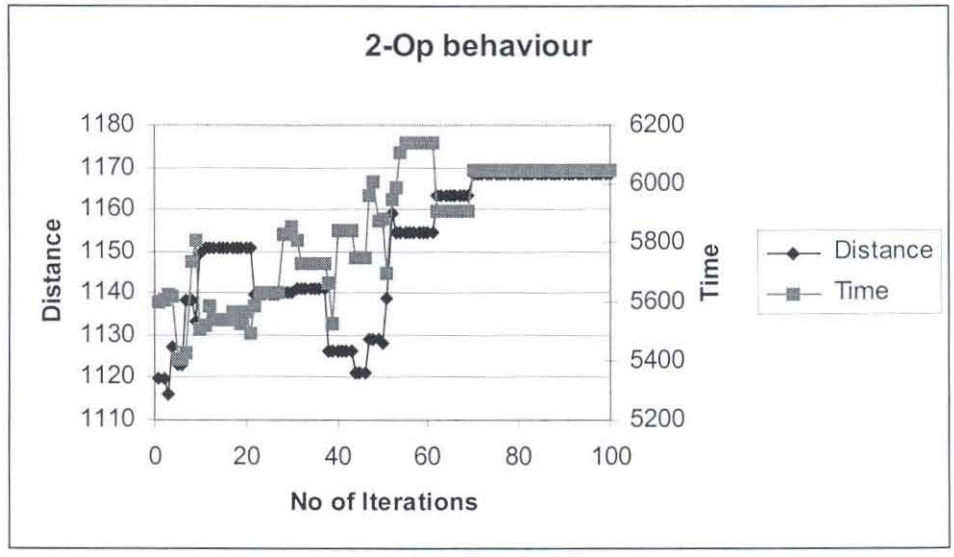

Figure 29: 2-Operator results

4.3. Application

In the previous paragraphs we showed the algorithm's performance with the 56 Solomon benchmark problems. This was done as a proof of concept for the algorithm. In this paragraph we will consider the result of a real life problem and show that the solution is feasible. The problem was taken from a commercial delivery company. All the variables were implemented as specified by the logistics manager.

Figure 30 shows the distribution of the stops as well as the solution. As stipulated in the initial research, the data environment is unpredictable. A quick analysis of the data indicates

- Inconsistent time window sizes.
- Random clustered stops.
- Long haul exceptions, relative to average stop distance from depot. The closest stop is less than 2 kilometres from the depot, while the furthest stop is more than 70 kilometres away.
- Some stops are located at the exact same position. From the figure we can make out some overlapping time windows.

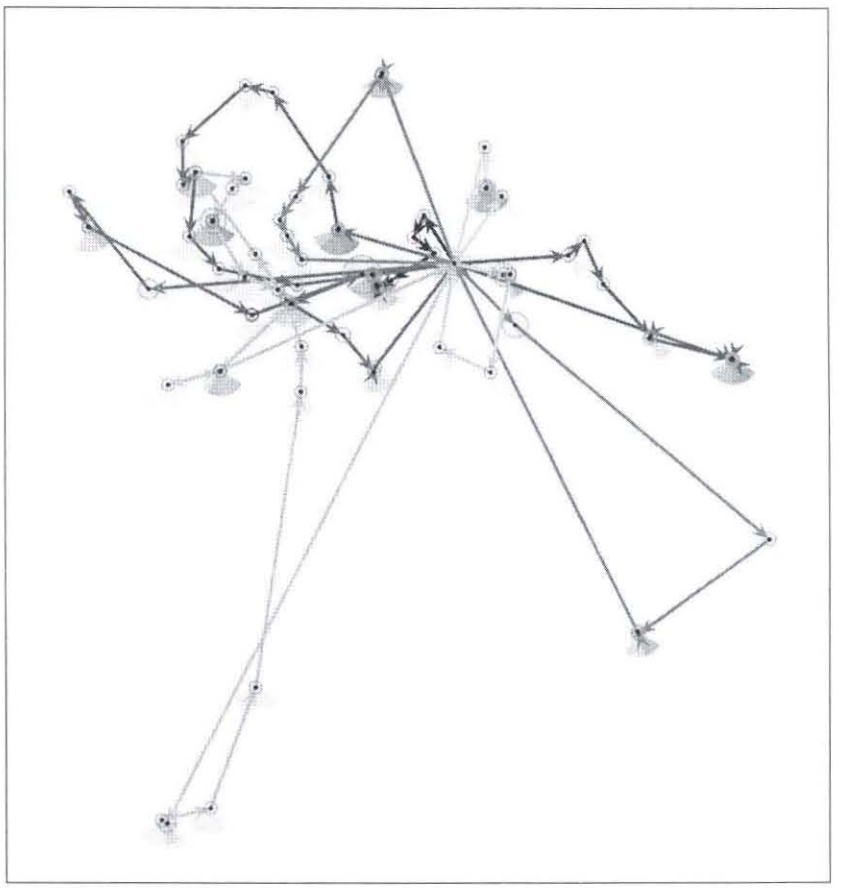

Figure 30: Application Solution 1

4.3.1. Initial Phase

From the results of initial solutions for Solomon's problems, we can conclude that using the latest departure time as criteria for a seed node will be sufficient. The following table shows the result of the initial solution on the real life problem. Because we are working with a heterogeneous fleet, all eight possible criteria have been implemented.

Initial Phase			
Criteria		Vehicles	Distance
Lowest Running Cost	Latest Departure	13	1251.92
	Earliest Arrival	12	1662.06
	Furthest	13	1259.07
	Closest	12	1186.82
Smallest Capacity	Latest Departure	13	1251.92
	Earliest Arrival	12	1522.54
	Furthest	13	1259.07
	Closest	12	1186.82

Table 8: Application Initial Phase

Although the latest departure criteria result in a comparative distance, the number of vehicles is higher than for the other methods. This confirms the decision to implement multiple criteria on the seed node selection.

4.3.2. Improvement

Figure 31 indicates the movement in the distance of the solutions for an execution of 5000 iterations. From the figure we can depict the ability of the algorithm to intensify and diversify.

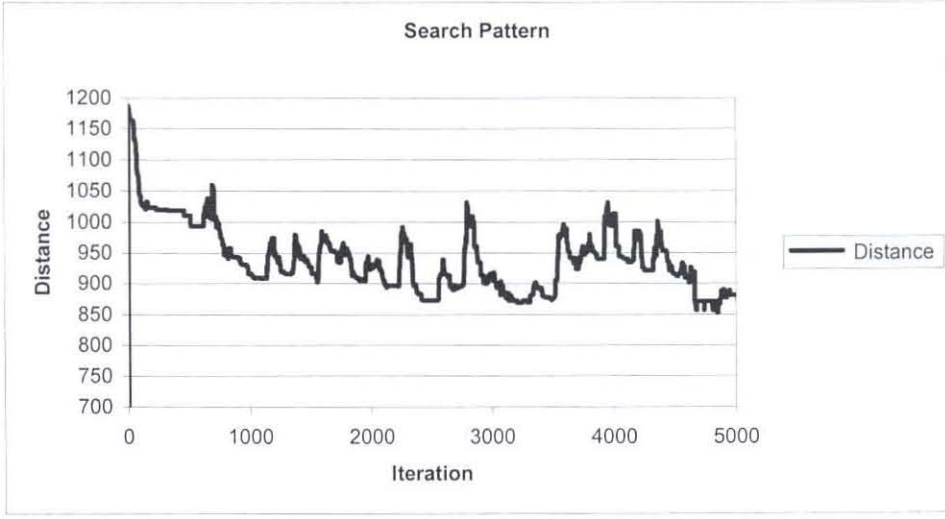

Figure 31: Search Pattern

The improvement heuristic started out with 12 vehicles and a distance of 1186 kilometres. After 5000 iterations we end up with 12 vehicles and a distance of 853 kilometres. This is an improvement of around 28% from the initial solution.

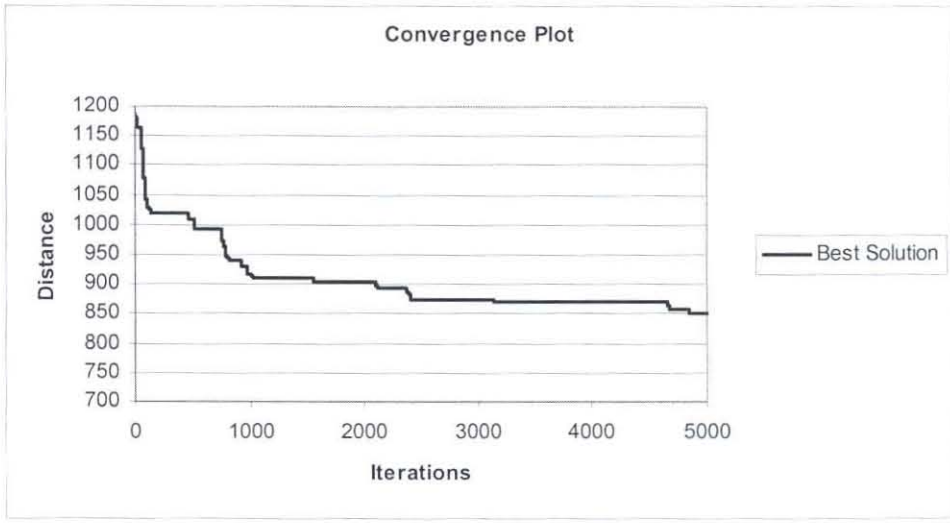

Figure 32: Convergence Plot

Another indication is the improvement in travel time. Travel time consist of the time it takes to travel between two stops depending on the time of the day. The travel time improved from 5789 minutes to 4497 minutes, an improvement of over 22% from the initial solution.

We also implemented multiple operations to move from a current solution to a valid neighbour solution. By keeping track of the success of an operation, we statistically balance the random selection of an operation. This technique results in the use of a better combination of operations depending on the data distribution and constraints of the problem instance. A hybrid with the Simulated Annealing method allows the solution to diversify and intensify periodically, while keeping track of moves through Tabu lists. Figure 31 indicates the ability of the algorithm to achieve this goal.

Figure 32 shows the ability of the algorithm to converge. We tested the 56 Solomon benchmark problems to indicate the validity of the algorithm. Solomon's problem is a simple instance of the problem we consider, but there does not exist benchmark problems for our set of problems. Table 5 shows that the new algorithm is effective on Solomon's benchmark problems.

The results prove that the implemented algorithm is effective to solve the set of problems encountered in an ASP environment. With the knowledge gained, we can continue to search for new operations and methods to improve the efficiency of the algorithm in the generic ASP environment.

BIBLIOGRAPHY

[1] Amberg, A., Domschke, W., Voß, S., 2000, Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees, European Journal of Operational Research 124, pp. 360-376

[2] Badeau, P., Gendreau, M., Guertin, F., Potvin, J., Taillard, E., 1995, A parallel tabu search heuristic for the vehicle routing problem with time windows, CRT-95-84, pp. 1-23

[3] Barbarosoglu, G., Ozgur, D., 1999. A tabu search algorithm for the vehicle routing problem. Computers and Operations Research, vol. 26, pp. 255-270.

[4] Bent, R.,van Hentenryck, P., 2001, A Two-Staged Hybrid Local Search for the Vehicle Routing Problem with Time Windows, Technical Report, Department of Computer Science, Brown University

[5] Bräysy, O., 2002, A Reactive Variable Neighborhood Search for the Vehicle Routing Problem with Time Windows, INFORMS Journal on Computing, pp. 1-22

[6] Bräysy, O., Dullaert, W., 2002, A Fast Evolutionary Metaheuristic for the Vehicle Routing Problem with Time Windows, Sintef Report, SINTEF Applied Mathematics, Research Council of Norway

[7] Bräysy, O., Gendreau, M., 2001, Metaheuristics for the Vehicle Routing Problem with Time Windows, Sintef Report, SINTEF Applied Mathematics, Research Council of Norway

[8] Bräysy, O., Gendreau, M., 2001, Route Construction and Local Search Algorithms for the Vehicle Routing Problem with Time Windows, SINTEF Applied Mathematics, Research Council of Norway

[9] Bräysy, O., Gendreau, M., 2001, Tabu Search Heuristics for the Vehicle Routing Problem with Time Windows, SINTEF Applied Mathematics, Research Council of Norway

[10] Bräysy, O., Hasle, G., Berger, J., Barkaoui, M. , 2003, Systematic Diversification Metaheuristic for the Vehicle Routing Problem with Time Windows, SINTEF Applied Mathematics, Department of Optimization

[11] Bräysy, O., Hasle, G., Dullaert, W., 2002, A Multi-Start Local Search Algorithm for the Vehicle Routing Problem with Time Windows, Preprint submitted to European Journal of Operational Research

[12] Bullnheimer, B., Hartl, R.F., Strauss, C., 1997, Applying the Ant System to the Vehicle Routing Problem, 2nd International Conference on Meta-heuristics, pp. 1-12

[13] Chin, A.J., Kit, H.W., Lim, A., 1999, A New GA Approach for the Vehicle Routing Problem, School of Computing, National University of Singapore

[14] Cordeau, J., Laporte, G., 2002, Tabu Search Heuristics for the Vehicle Routing Problem, Les Cahiers du GERAD

[15] Cousineau-Ouimet, K., 2002, A Tabu Search Heuristic for the Inventory Routing Problem, Department of Quantitative Methods, École des Hautes Études Commerciales

[16] Crispim, J., Brandão, J., 2001, Reactive Tabu Search and Variable Neighbourhood Descent Applied to the Vehicle Routing Problem with Backhauls, MIC'2001 - 4th Metaheuristics International Conference

[17] De Backer, B., Furnon, V., Kilby, P., Prosser, P., Shaw, P., 1997, Solving Vehicle Routing Problems using Constraint Programming and Metaheuristics, Journal of heuristics

[18] De Backer, B., Furnon, V., 1997, Meta-heuristics in Constraint Programming Experiments with Tabu Search on the Vehicle Routing Problem, 2nd International Conference on Metaheuristics

[19] De Klerk, H.F., 2001, The investigation and formulation of generic algorithms for decision support systems of order fulfilment and distribution, Thesis, University of Pretoria

[20] Doerner, K., Hartl, R.F., Reimann, M., 2001, A hybrid ACO algorithm for the Full Truckload Transportation Problem, Technical Report, University of Vienna, Austria

[21] Dullaert, W., Bräysy, O., 2003, Routing Relatively Few Customers per Route, Top, Volume 11, Number 2, pp. 325-336

[22] Duncan, T., 1995, Experiments in the use of Neighbourhood Search techniques for Vehicle Routing, Artificial Intelligence Applications Institute, University of Edinburgh

[23] Hasle, G., 2003, Heuristics for Rich VRP Models, Presentation, Department of Optimization, SINTEF Applied Mathematics, University of Oslo

[24] Havenga, A., 2001, Investigation And Definition Of Market And User Requirements For The Development Of An Order Fulfillment Engine, Thesis, University of Pretoria

[25] Hertz, A., Taillard, E., de Werra, D., 1995, A tutorial on tabu search, EPFL, Département de Mathématiques.

[26] Hertz, A., Widmer, M., 2003, Guidelines for the use of meta-heuristics in combinatorial optimization, European Journal of Operational Research, Volume 151, pp. 247-252

[27] Ho, W., Chin, J., Lim, A., 1998, A Hybrid Search Algorithm for the Vehicle Routing Problem with Time Windows, International Journal on Artificial Intelligent Tools, pp. 1-19

[28] Ho, S.C., Haugland, D., 2002, A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows and Split Deliveries, Department of Informatics, University of Bergen

[29] Kilby, P., Prosser, P., Shaw, P., 1997, Guided local Search for the Vehicle Routing Problem, 2nd International Conference on Metaheuristics

[30] Kilby, P., Prosser, P., Shaw, P., 1999, A Comparison of Traditional and Constraint-based Heuristic Methods on Vehicle Routing Problems with Side Constraints, Kluwer Academic Publishers, Boston, pp. 1-23

[31] Kulturel-Konak, S., Norman, B. A., Coit, D. W., Smith, A. E., 2001, Exploiting tabu search memory in constrained problems, INFORMS Journal on computing

[32] Laguna, M., Glover, F. 2004, What is Tabu Search, http://www.tabusearch.net/Tabu_Search/What_is_Tabu_search.ASP, visited on 22 March 2004

[33] Laporte, G., 1999, Classical and modern heuristics for the vehicle routing problem, Internal report, Centre for Research on Transportation, Montreal.

[34] Larsen, J., 1999, Vehicle Routing with Time Windows - Finding optimal solutions efficiently, pp. 1-15

[35] Lau, H.C., Liang, Z., 2000, Pickup and Delivery with Time Windows : Algorithms and Test Case Generation, School of Computing, National University of Singapore

[36] Li, H., Lim, A., 2001, A Metaheuristic for the Pickup and Delivery Problem with Time Windows, Department of Computer Science, National University of Singapore

[37] Mester, D., Bräysy, O., 2003, Guided Evolution Strategies for Large Scale Vehicle Routing Problem with Time Windows, Institute of Evolution, Mathematical and Population Genetics Laboratory, University of Haifa

[38] O'Rourke, K.P., Glenn Bailey, T., Hill, R., Carlton, W.B., 1999, Dynamic Routing of Unmanned Aerial Vehicles Using Reactive Tabu Search, 67th MORS Symposium, Working Group 10 -- Unmanned Systems, pp. 1-41

[39] Qili, Z., 1999, Heuristic Methods For Vehicle Routing Problem with Time Windows, Thesis, Department of Electrical Engineering, National University of Singapore

[40] Ombuki, B.M., Nakamura, M., Osamu, M., 2002, A Hybrid Search Based on Genetic Algorithms and Tabu Search for Vehicle Routing, Technical Report, CS-02-07, pp. 1-7

[41] Rego, C., 1996, A Subpath Ejection method for the Vehicle Routing Problem, Departamento de Informatica, Universidade Portucalense

[42] Rego, C., 1997, Node Ejection Chains for the Vehicle Routing Problem: Sequential and Parallel Algorithms, Departamento de Informatica, Universidade Portucalense

[43] Rochat, Y., Taillard, E.D., 1995, Probabilistic diversification and intensification in local search for vehicle routing, Journal of heuristics, Volume 1, pp. 147-167

[44] Ryer, D.M., 1999, Implementation Of The Metaheuristic Tabu Search In Route Selection For Mobility Analysis Support System, Thesis, AFIT/GOA/ENS/99M

[45] Solomon, M., Solomon's Solutions, www.cba.neu.edu/~solomon/poroblems.html, visited 22 March 2004

[46] Taillard, E.D., 1996, A heuristic column generation method for the heterogeneous fleet VRP, CRT-96-03, pp. 1-13

[47] Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Potvin, J., 1996, A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time

Windows, Report, Centre de recherche sur les transports, Université de Montréal

[48] Taillard, E.D., Gambardella, L.M., Gendreau, M., Potvin, J., 2001, Adaptive memory programming: A unified view of metaheuristics, European Journal of Operational Research 135, pp. 1-16

[49] Taillard, E.D., Laporte, G., Gendreau, M., 1995, Vehicle Routing With Multiple Use Of Vehicles, CRT-95-19, pp. 1-13

[50] Tan, K.C., Lee, L.H., Zhu, K.Q., 1999, Heuristic Methods for Vehicle Routing Problem with Time Windows, National University of Singapore

[51] Tansini, L., Urquhart, M., Viera, O., 2000, Comparing assignment algorithms for the Multi-Depot VRP, Dpto. Investigación Operativa, Instituto de Computación, Facultad de Ingeniería, UDELAR.

[52] Van Schalkwyk, W.T., 2002, An algorithm for the Vehicle Routing Problem with various side constraints, Thesis, University of Pretoria

[53] Winston, W.L., 1994, Operations Research: Applications and Algorithms, Third edition, California.

[54] Xu, J., Kelly, J.P., 1996, A Network flow-based Tabu Search heuristic for the Vehicle Routing Problem, Thesis, Graduate School of Business, University of Colorado at Boulder