# Development of a Scheduling Model and Solution Algorithms for Carriers in the Automotive Manufacturing Environment

J.D. Marx

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Industrial Engineering at the University of Stellenbosch.

Prof. W. van Wijck

Mr. J. Bekker

APRIL 2006

# Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: Date: 6 March 2006

# Summary

The aim of this thesis was to develop a scheduling model and solution algorithms for a generic problem that can be used to solve a scheduling problem at Autocarriers.

The main characteristics of the generic problem are the following: Transporters are required to deliver various containers (each having a specific origin vertex and destination vertex) by transporting them through a network. Each transporter has a restricted cargo capacity which is a function of the number and types of containers loaded on the transporter. Containers can be stored temporarily at any vertex so that they can be moved by one or more transporters. Costs are incurred by delivering containers later than the required time and for transporters moving along the arcs of the network.

A problem definition was created that describes the various entities of the generic problem. The problem definition was used to develop a scheduling model consisting of an objective function and a solution space. The solution space is defined by a set of conditions that describes all valid solutions to the problem.

Two heuristic algorithms were developed to generate solutions to the problem. The maximum service level algorithm attempts to deliver containers on time without any regard to the cost involved. The second algorithm, the minimum total cost algorithm, attempts to minimise the sum of the costs of the problem.

A software tool was developed that can be used to define an instance (or specific occurrence) of the generic problem and solve the problem approximately by means of the solution algorithms. The software tool has a simulation capability that is useful for evaluating the solution algorithms; hence various changes to a problem can be evaluated.

# Opsomming

Die doel van hierdie projek was om 'n skeduleringsmodel en oplossingsalgoritmes te ontwikkel vir 'n generiese problem. Hierdie model kan gebruik word om 'n skeduleringsprobleem by Autocarriers op te los.

Die belangrikste kenmerke van die generiese probleem kan soos volg uiteengesit word: Vervoeroperateurs word benodig om verskeie houers deur 'n netwerk te vervoer. Elke houer het 'n spesifieke oorsprongsnode en bestemmingsnode. Die vervoeroperateurs het 'n beperkte laaikapasiteit wat afhanklik is van die aantal houers van elke tipe wat opgelaai word. Die houers kan tydelik by enige node gestoor word sodat hulle deur een of meer vervoeroperateurs vervoer kan word. Kostes word toegeken vir die laat aflewering van houers sowel as vir die beweging van vervoeroperateurs deur die netwerk.

'n Probleemdefinisie is ontwikkel wat die verskillende entiteite in die probleem omskryf. Die probleemdefinisie is gebruik om 'n skeduleringsmodel te ontwikkel wat bestaan uit 'n doelwitfunksie en 'n oplossingsruimte. Die oplossingsruimte is gedefinieer deur 'n stel voorwaardes wat alle geldige oplossings beskryf.

Twee heuristiese algoritmes is ontwikkel om oplossings vir die probleem te genereer. Die maksimum diensvlak algoritme het die betydse aflewering van houers ten doel, sonder in agneming van die gepaardgaande koste. Die tweede algoritme, die minimum totale koste algoritme, poog om die som van die verskeie kostes van die probleem te minimeer.

'n Sagtewarepakket is ontwikkel wat gebruik kan word om 'n probleemgeval te definieer wat in ooreenstemming is met die generiese probleem. Die sagtewarepakket kan hierdie probleem met behulp van die oplossingsalgoritmes oplos. 'n Simulasiefunksionaliteit is in die sagtewarepakket aangebring sodat die algoritmes evalueer kan word. Gevolglik kan die invloed van verskeie veranderinge aan die probleemgeval ondersoek word.

# Acknowledgements

The author would like to thank the following people for their contribution to the project:

Mr. Chris Husted from Autocarriers for help in understanding the Autocarriers problem and for providing access to the company's data.

Prof. Willie van Wijck, for his guidance, advice and moral support.

Mr. James Bekker, for his guidance and help, especially in the area of discrete event simulation.

My wife, Kerry, for her loving support in this project.

# Table of Contents

# List of Figures

# List of Tables

# Glossary

| | |
|---|---|
| $\delta$ | A scale factor for container penalties in the MTC algorithm |
| ACO | Ant Colony Optimisation |
| Arc | A directional connection between two vertices having a certain length. Represents an allowable movement of the transporters and containers. |
| AS | Ant System |
| ASCII | American Standard Code of Information Interchange |
| Average Container Late Time | The average time that a container can be expected to be late, given that it will be late |
| CDVRP | Capacity and Distance restricted Vehicle Routing Problem |
| $C_n$ | The cost of routing a container with transporter $n$ in the MTC algorithm |
| Container | A unit size cargo representing an entity that must be transported in the system |
| Container Template | A family or category of similar containers |
| $C_p$ | The penalty cost incurred by a single container in the MTC algorithm |
| $C_r$ | The running cost of a transporter in the MTC algorithm |
| CVRP | Capacitated Vehicle Routing Problem |
| Delay Time | The time required by a transporter to enter a vertex, unload, reload, refuel and exit the vertex |
| Destination Time | Latest acceptable delivery time of a container. If a container is delivered later than its destination time a penalty is incurred. |
| DVRP | Distance Constrained Vehicle Routing Problem |
| Edge | A non-directional connection between two vertices |
| FSMVRP | Fleet Size and Mix VRP (same as HVRP) |
| FSMVRP | Fleet Size and Mix VRP with Time Windows |
| GA | Genetic Algorithm |
| Generic Problem | A generalised problem that can be solved with the scheduling model and solution algorithms developed in this project |
| HFFVRP | Heterogeneous Fixed Fleet VRP |
| HVRP | Heterogeneous Fleet VRP (same as FSMVRP) |

| | |
|---|---|
| ILP | Integer linear programming problem |
| $L_c$ | The list of undelivered containers |
| LP | Linear programming problem |
| Loading Rule | A single rule used to describe the cargo capacity restriction of a transporter |
| Log | Natural Logarithm |
| $L_T$ | The list of transporters available in a period |
| MDVRP | Multiple-Depot Vehicle Routing Problem |
| MDVRPTW | Multiple-Depot Vehicle Routing Problem with Time Windows |
| Movement | A directional movement of a transporter on an arc between two cities with which containers can be associated |
| MSL | The maximum service level scheduling algorithm |
| MTC | The minimum total cost scheduling algorithm |
| m-VRPTW | VRP with Time Windows and a fixed fleet |
| OVRP | Open Vehicle Routing Problem |
| Problem Instance | An instance of the generic problem defined in this project |
| RRVRP | Rollon-Rolloff Vehicle Routing Problem |
| SA | Simulated Annealing |
| SDVRP | Split Delivery Vehicle Routing Problem |
| Service Level | The ratio of containers delivered on time to the total number of containers |
| Slack Time | The time window between when a container is available for pickup and the its destination time |
| Solution Space | A framework of conditions which describes all valid scheduling solutions of a problem instance |
| SVRP | Stochastic Vehicle Routing Problem |
| System | All the entities of a problem instance |
| T | The system time when an algorithm is executed |
| $t_c$ | The last arrival time of a container, which is the time it will next be available for pickup by a transporter. Used in the MSL algorithm. |
| Template | A family of type of entities. See *container template* and *transporter template.* |
| Total Cost | The sum of the penalty and running costs of a solution |
| Total Penalty Cost | The sum of penalties paid for the late delivery of containers |

| | |
|---|---|
| Total Running Cost | The sum of running costs incurred by all moving transporters |
| Transportation Network | The graph made up of vertices and arcs representing all allowable locations and movements of transporters and containers |
| Transporter | A carrier entity in the system |
| Transporter Efficiency | A percentage reflecting how efficiently transporters have been utilised for transporting containers, weighed according to the running cost |
| Transporter Template | A family or category of similar transporters |
| TS | Tabu Search Algorithm |
| Vertex | A representation of a physical location in a transportation network |
| VRP | Vehicle Routing Problem |
| VRPB | VRP with Backhauls (See VRPPD) |
| VRPPD | Vehicle Routing Problem with Pickup and Deliveries |
| VRPTW | Vehicle Routing Problem with Time Windows |

# Chapter 1 Introduction

This chapter contains a short background to the project and describes the objectives of the thesis. Motivation is given for the development of a more general problem description and a description of the document layout is then given.

## 1.1. BACKGROUND OF THE PROJECT

The project originated from a scheduling problem at Autocarriers concerning the scheduling of vehicle carriers. Autocarriers transports vehicles for most of the major automotive manufacturing companies in South Africa.

The manufacturing of vehicles is initiated by dealer orders placed at the plants. Some plants can manufacture the vehicles in the same sequences as the placed orders, since they have flexible assembly lines which allow different types of vehicles to be assembled without changeovers. In other cases, vehicles are manufactured in batches of similar vehicles. In either case the completed vehicles are placed in a temporary storage area at the plant.

Once a vehicle is placed in the storage area of a plant Autocarriers is notified by an automated information system that the vehicle is ready for pickup. Autocarriers is then given the exact details of the vehicle, such as its dealer destination, latest delivery time, type and serial number. If Autocarriers fails to deliver a vehicle before its latest delivery time, a penalty cost is incurred for each day the vehicle is late. The time frame between the release of a vehicle and its latest delivery time, and the penalty cost vary amongst the manufacturers. Some manufactures aid Autocarriers' scheduling by notifying them in advance about the number and types of vehicles that will be manufactured, but the information is only finalised once the vehicle is placed in the storage area.

Autocarriers are contracted to transport the vehicles from the storage area to the relevant dealers. Consequently, carriers travel between all major cities and towns in Southern Africa.

Autocarriers has established depots in the major cities where they can store vehicles temporarily. In many cases the orders the dealers place at the manufacturers are large enough that Autocarriers can allocate a carrier to a single dealer order so that the carrier fetches the vehicles from the temporary storage area of the plant and transports them directly to the dealer. When an order is smaller, it becomes less economical to allocate a truck to only one dealer order. In these situations the vehicles are picked up from the

temporary storage area and moved to Autocarriers' depot in the vicinity of the plant. The vehicles are then dispatched form the depot near the plant to the closest depot to the dealer. The vehicles are then unloaded and reloaded onto another carrier and taken to their destinations. It is also possible for the vehicles to be stored at intermediate depots while they are moved from the depot at the manufacturer to the depot near the dealer.

The transportation of vehicles is accomplished by means of specialised carriers which contain slots of various types onto which vehicles can be loaded. Certain vehicles can only be loaded onto certain carrier slots. The Mercedes Benz A-Class is, for example, too high to be loaded onto the lower decks of the carriers. Autocarriers owns nearly a hundred carriers of various types. Each carrier type has different number of slots, slot configurations, travelling speeds and travelling costs.

The scheduling of the carriers is currently done at Autocarriers' depot in Bellville. The managers at the different depots fax or email the number and types of vehicles that require transportation, as well as information regarding the availability of the carriers that are currently at the depot. This information is then fed into a confidential algorithm that develops an initial schedule for the carriers. The initial schedule is then improved and finalised by a human scheduler. The schedule for each carrier is then emailed to the depot where its precious route ended and handed over to the next assigned driver.

The carriers are fitted with GPRS and GPS devices so that the vehicle can be tracked. Autocarriers is contractually obliged to fit the carriers with GPS devices so that a manufacturer can request the position of a vehicle at any time. Once a vehicle is picked up or delivered by a carrier, the serial number of the vehicle is relayed to the Bellville depot by means of the GPRS device fitted to the carrier. This enables Autocarriers to monitor whether a carrier is picking up the correct vehicles and whether the correct vehicles are delivered at the dealers (or depots). The GPRS information also allows Autocarriers to monitor the carrier schedule, since the service time of the carrier at the dealer (or depot) can be compared with the estimated times of the schedule. The drivers are issued with mobile phones and can thus be contacted if they deviate from the carrier schedule.

The drivers are scheduled according to the requirements of the carrier schedule. The carriers contain small sleeping compartments which enable the carriers to travel long distances without changing drivers. South African labour and traffic regulations require the drivers to rest a certain number of hours every day. Consequently, the route duration of schedules must be adapted when single drivers are assigned to the carriers. Alternatively, two drivers

can be assigned to a carrier, allowing 24 hour operation. Autocarriers does not regard driver availability and scheduling as current constraints on the scheduling of carriers and vehicles. The driver availability and scheduling was therefore not considered in this project.

Autocarriers is continually faced with various questions concerning the scheduling of their carriers. These questions arise from possible changes to their current setup. It is, for example, difficult to determine what the profit or loss from a specific customer is, since the carrier travelling costs of the different customers are interrelated. It is also difficult to determine whether a different scheduling philosophy would result in higher profits. In addition, Autocarriers can change their carrier fleet or add depot facilities, but the effects of these alterations cannot currently be determined. The manufacturers regularly offer new contracts and Autocarriers requires the ability to determine the break-even point for these offers.

This project involved the development of a software tool and scheduling model with solution algorithms that can be used to examine the aforementioned aspects of Autocarriers' operation, especially with regard to the transportation of vehicles between their different depots.

It was decided that the Autocarriers problem can be used as a basis for defining a more general scheduling problem. The motivation for generalising the problem is given in the next section.

## 1.2. GENERALISATION AND UNIVERSALITY OF THE PROBLEM

The Autocarriers problem was used as a basis for defining a more general problem (referred to in this document as the generic problem). The main motivation for considering a more general problem is that the operating environment of Autocarriers is changing continually. Examples of changes to the operating environment include new labour law regulations, changes to the carrier fleet and types of carriers, and changes to their business objectives. It is therefore important to describe the problem in more general terms so that it can be used as a basis to solve a variety of problems similar to the current scheduling problem at Autocarriers.

The basic characteristics of a problem that can be mapped on the generic problem are as follows: Containers of various types require transportation from specific origins to specific destinations. There are a variety of transporters available to transport the containers, each having a finite cargo capacity that is dependent on the quantity and types of containers loaded on the transporter. The containers can be stored temporarily at the various locations to be picked up later and carried further by other transporters. More than one transporter can thus be used to move a single container. The transporters do not necessarily move directly from the origin location of the container to the destination location, since the transporter may move through various intermediate locations.

The problem has two cost factors: the travelling cost of the transporters and the penalty cost incurred by delivering containers later than a requested time. Delivery time frames (or windows) and penalty rates of the various containers may differ. Similarly, the running costs of the various transporters may differ.

Many instances of the generic problem may exist. An example of another instance of the generic problem is the scheduling of cargo on ships. Ships, by their nature, have limited cargo capacities. Cargo can be classified in various categories and consequently the cargo capacity restriction of a ship can be modelled as a function of the number of containers of each category carried by the ship. The containers can be stored temporarily at the different harbours and carried further by other ships. Another example of a problem that can be mapped on the generic problem is transporting people by buses. The only limitation on the cargo capacity of a bus is the number of people that can be transported. There is only one container type in this problem, namely a passenger. A model can thus be developed to transport people by means of a variety of buses from their pickup locations to their drop-off locations. Like the shipping problem, a cargo scheduling problem involving aircraft can also be described in terms of the generic problem.

## 1.3. PROJECT OBJECTIVES

The project objectives were the following:

a) to define a generic problem that can be used to describe a variety of problems including the Autocarriers problem;
b) to develop a scheduling model with solution algorithms that can model and solve the Autocarriers problem approximately;
c) to develop a software tool that can house the scheduling model and apply the solution algorithms; and
d) to give the software tool a simulation capability so that various configurations of a problem instance can be examined.

The requirements of the software tool were:

a) The software tool must be user-friendly.
b) The software tool must be able to handle problems of realistic size and complexity so that it can be used to solve real-world problems, specifically the Autocarriers problem.

## 1.4. CHAPTER SUMMARY AND DOCUMENT LAYOUT

The background of the project and the main characteristics of the generic problem were discussed in this chapter. The project objectives were stated as well as the requirements of the software tool.

Chapter 2 contains a literature study describing the models and methods currently used for solving similar problems.

Chapter 3 contains a detailed definition of the generic problem and a description of the scheduling model developed. The model consists of a chosen objective function and solution space. The underlying assumptions of the scheduling model are also described in this chapter.

The conceptual design of the software tool is described in chapter 4. The tool consists of five main program structures and a database structure. The program structures make it possible to define an instance of the generic problem, create a solution to the problem (either manually or automatically) and evaluate the solution. There are also structures that make it possible to simulate various configurations of a problem instance, and to analyse the input and output of the simulations. The database structure stores the problem instance, solution obtained and simulation setup.

Chapter 5 contains a detailed discussion of the development of the tool, and the solution algorithms are described in detail. The two solution algorithms developed are a heuristic to maximise the service level of a solution, and a second heuristic to minimise the total cost of a solution.

Chapter 6 contains a short case study of the Autocarriers problem, showing how the scheduling model, solution algorithms and software tool can be used to solve a real-world problem and get answers to questions raised at Autocarriers (mentioned earlier in this chapter).

In chapters 7 and 8 the respective conclusions and recommendations resulting from the thesis project are stated.

Lastly, chapter 9 contains a summary of the project.

# Chapter 2 Literature Study

The vehicle routing problem (VRP) was introduced by Dantzig and Ramser [1] in 1959 as the "truck dispatching problem" and has become the most prominent member in the distribution routing problem arsenal. This chapter contains a survey of current vehicle routing models and algorithms available, as well as a short description of practical implementations of vehicle routing problems.

## 2.1. VEHICLE ROUTING PROBLEM VARIATIONS

The vehicle routing problem is a name of a set of problems in which a set of routes for a fleet of vehicles must be found from one or more depots to various customers.

Many variations of the VRP have been defined. The definitions of these models are not cast in stone and often differ in the literature. In this section a short description is given of a variety of VRP models and their characteristics.

### 2.1.1. The classic VRP

The objective of the VRP [2] is to deliver goods to a set of customers with known demands by using a homogeneous fleet of vehicles, travelling from a single depot to customers and returning to the depot at a minimum cost.

The classic (or basic) VRP can be described as a graph theory problem as follows:

- Let $G = (V, A)$ be a complete directed graph where $V$ is the vertex set and $A$ an arc set.

  In the vertex set $V = \{v_0, v_1, v_2, \ldots, v_n\}$, $v_0$ is a depot, while the remaining vertices, $v_i$ ($i \geq 1$), are customers that are associated with a non-negative demand $q_i$ and a service time $s_i$.

  $A$ describes the arc set connecting the vertices so that $A = \{(v_i, v_j): v_i, v_j \in V, i \neq j\}$. (An arc is a directional connection between two vertices.)
- Let $C$ be a cost matrix that is associated with $A$ so that $c_{ij}$ is the nonnegative cost of traversing arc $(v_i, v_j)$. Where $i = j$ we set $c_{ij} = \infty$.
- Let $T$ be a transit time matrix that is associated with $A$ so that $t_{ij}$ is the transit time of arc $(v_i, v_j)$. Where $i = j$, we set $t_{ij} = 0$.
- Let $T_{max}$ be the preset maximum time allowed for visiting all vertices on a vehicle route.

- If $c_{ij} = c_{ji}$, $t_{ij} = t_{ji}$ $\forall (v_i, v_j) \in A$, then it is common practice to replace the arc set with an edge set $E = \{(v_i, v_j): v_i, v_j \in V, i < j\}$. (An edge is a non-directional connection between two vertices.)
- Let $m$ be the size of the homogeneous vehicle fleet. It can either be known in advance or used as a decision variable. Each vehicle can be assigned to only one vehicle route (referred to in some texts as a circuit).

The classic VRP consists of finding $m$ vehicle routes (one for each vehicle) of minimal total cost such that:

1. Each vehicle route starts and ends at the depot, $v_0$.
2. Each vertex $v_j \in V\backslash\{v_0\}$ is visited exactly once by exactly one vehicle route.
3. The total duration of each vehicle route (including the travel and service times) does not exceed the preset limit $T_{max}$. (This condition is often discarded.)

### 2.1.2. The capacitated VRP (CVRP)

In the capacitated vehicle routing problem (CVRP) [2] a homogeneous fleet of vehicles with restricted capacity, $Q$, must deliver goods from a single depot to various customers with known demands. The classic VRP becomes a CVRP when the following restriction is added:

1. The sum of the demands of the vertices visited in a circuit may not exceed the vehicle capacity $Q$.

If the cost or travel time matrices are not symmetrical the CVRP is referred to as an asymmetrical CVRP (or ACVRP), otherwise it is referred to as a symmetrical CVRP (or SCVRP).

### 2.1.3. The distance constrained VRP (DVRP) and capacitated DVRP (CDVRP)

A limit may be imposed on the distance traversed by each vehicle of the classic VRP. In other words, a vehicle route may not exceed a certain length. The resulting problem is the distance constrained VRP (or DVRP) [2]. The DVRP is useful for describing problems where the vehicle may only refuel at the depot ($v_0$). The vehicles are consequently limited by the distance they can travel on a single tank of fuel.

If $D_{max}$ is the maximum allowable distance of a vehicle route and $d_{ij}$ is the distance of $(v_i, v_j) \in A$, the classic VRP can be transformed to a DVRP by adding the following constraint:

1. The sum of all $d_{ij}$s in a vehicle route may not exceed $D_{max}$.

The distance restriction can also be imposed on the capacitated vehicle routing problem, resulting in the capacity and distance restricted VRP (or CDVRP) [3]. The distance restriction is sometimes implicitly assumed in texts. As a consequence, the DVRP and CDVRP are sometimes simply referred to as the VRP and the CVRP respectively.

### 2.1.4. The VRP with time windows (VRPTW)

The VRP with time windows [4] results from customers imposing restrictions on the delivery time of goods. Two cases are defined:

a) In the hard case a vehicle must service $v_i$ in a time window starting at the earliest time $e_i$ and closing at the latest time $l_i$. If the vehicle service time at the vertex, $b_i$, does not fall within this window, the solution is regarded as infeasible.
b) In the soft case, however, the windows may be violated, but a penalty is incurred.

Figure 2.1 below illustrates how the time window can be represented on a timeline.

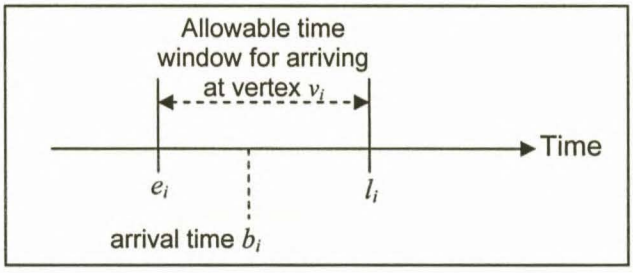



*Figure 2.1 Illustration of a delivery time window*

The following constraint is added to the VRP definition to impose hard time windows:

1. The service of vertex $v_i$ must start after $e_i$ and end before $l_i$.

Two scenarios exist when soft time windows apply:

1. Firstly, only the service deadline may be violated at a cost and by a length of time limited by $l_i'$. This results in an enlarged time window starting at $e_i$ and ending at $l_i + l_i'$.
2. Secondly, both time windows can be violated at a cost. As in the first case, the deadline may be violated by a length of time limited by $l_i'$. Similarly, the earliest start time can be violated at a cost and a length of time limited by $e_i'$, resulting in the enlarged time window starting at $e_i - e_i'$ and ending at $l_i + l_i'$.

If a time window is defined for the depot (called a *scheduling horizon*) every circuit that includes the depot must start and end within the scheduling horizon.

In the VRPTW it is often assumed that the vehicles have limited capacity and distance, although this is not explicitly denoted in the name VRPTW. Some researchers make these constraints explicit by referring to the capacitated and distance restricted VRPTW or CDVRPTW.

### 2.1.5. The multiple-depot VRP (MDVRP) and MDVRPTW

The multiple-depot VRP (MDVRP) describes the case where a company has more than one depot from which it can serve the various customer demands. A fleet of vehicles is stationed at each depot. Consequently, a vehicle travels from its base depot on a vehicle route to one or more customers and returns to its base depot. The aim of the MDVRP is to serve all customer demands from the various depots while minimising the vehicle fleet size and transit cost incurred.

The multiple-depot VRP with time windows (MDVRPTW) [5] is a combination of the MDVRP and VRPTW problems. In the MDVRPTW, time windows are imposed on the servicing of customers and a scheduling horizon can be defined for each depot.

The MDVRP and MDVRPTW are often solved by clustering (or assigning) the customers to a depot and then routing the vehicles at each depot through the customers assigned to the depot. Many "cluster first; route second" algorithms have been developed for the MDVRP.

### 2.1.6. Vehicle routing problems with fleet constraints

A couple of VRP models that occur less frequently in the literature impose certain constraints or criteria on the vehicle fleet. Some of these problems are worth mentioning in the context of this thesis.

(a) The VRP with time windows and a limited number of vehicles (m-VRPTW)

Most vehicle routing problems assume that a sufficiently large number of vehicles are available in the fleet. The aim of these problems is then to minimise the number of vehicles used and the routing cost. Lau et al. [6] defined the VRP with time windows and a limited number of vehicles (m-VRPTW) in which customers require service within a certain time window, and a limited homogeneous fleet is available to serve the customers. They specifically examined problems that are over-constrained, that is to say, problems where the limited fleet may make it impossible to serve all the customers in the required time windows.

(b) The heterogeneous fleet VRP (HVRP or FSMVRP)

The vehicle fleet in vehicle routing problems is often assumed to be homogeneous, that is to say, the vehicles have same capacity, speed and transit cost. In reality, companies usually deploy a variety of vehicles to meet different demands and costs.

The heterogeneous fleet vehicle routing problem (HVRP) is the result of relaxing the assumption of a homogeneous fleet. The heterogeneous fleet VRP is sometimes referred to as the *fleet size and mix VRP* (FSMVRP) [7] or the *fleet size and composition VRP*.

In the FSMVRP each vehicle is allowed to have a different acquisition cost, transit cost, speed and capacity. The aim of the FSMVRP is to minimise the sum of acquisition and transit costs, subject to the adapted constraints of the VRP. (The vehicle capacity constraint of the VRP must be adapted for each vehicle.)

One of the most effective heuristic algorithms developed for solving the FSMVRP is the tabu search of Gendreau et al. [8].

(c) The fleet size and mix VRP with time windows (FSMVRPTW)

The fleet size and mix VRP with time windows (FSMVRPTW) [9] is a combination of the VRPTW and FSMVRP. Time windows are imposed on serving the customer demands, and a scheduling horizon can optionally be imposed on the depot. The time windows can either be hard or soft, as was the case with the VRPTW. The aim of the FSMVRPTW is to serve the customers within the imposed time window limits at a minimum sum of vehicle routing and acquisition cost.

The FSMVRP and FSMVRPTW problems have been solved using savings-based algorithms (especially with the Golden et al. [10] adaptations), matching-based savings, "route first, cluster second" and tabu search algorithms. Route improvement algorithms have also been applied to the solutions found by these algorithms.

(d) The heterogeneous fixed fleet VRP (HFFVRP)

When the fleet size in the heterogeneous fleet VRP is assumed to be finite the problem is referred to as the heterogeneous fixed fleet VRP or HFFVRP [11]. The aim of the HFFVRP is to minimise the routing cost of the different vehicles while the acquisition cost is disregarded. The HFFVRP has been solved efficiently with deterministic annealing.

### 2.1.7. The VRP with pickups and deliveries (VRPPD)

An interesting variation of the vehicle routing problem is the VRP with pickups and deliveries (VRPPD) [12], in which a heterogeneous set of vehicles (which are based at multiple terminals) must complete a number of transportation requests. Each request consists of a pickup point, a delivery point and a volume to be transported between these points. An important aspect of the VRPPD is that the goods do not move through a depot. The dial-a-ride problem is a well known example of a VRPPD.

Several constraints are imposed on the VRPPD which include: visiting each pickup and delivery only once, vehicle capacity restrictions, and restrictions ensuring that a pickup happens before its associated delivery and that the pickup and delivery happens on the same vehicle route. Depot constraints are imposed to ensure that a vehicle returns to its terminal, and resource restrictions may limit the number of vehicles.

The VRPPD with time windows (VRPPDTW) is a generalisation of the VRPTW, which makes pickups and deliveries possible. Practical applications of the VRPPTW include the transport of people and the air lift of cargo.

### 2.1.8. The VRP with backhauls (VRPB)

An extension of the capacitated VRP is the VRP with backhauls (VRPB), where the set of customers is divided into linehaul customers, each requiring a delivery of goods, and backhaul customers, each requiring a pickup of goods by the vehicles.

As was the case with the capacitated VRP, each vehicle route must start and end at the depot, each vehicle performs one route and the vehicle capacity may not be exceeded. The objective function in these problems is usually to minimise the total distance travelled by the vehicles (and therefore the total cost).

A precedence constraint is usually imposed on the vehicle routes so that the backhaul customers are visited after the linehaul customers. The constraint is motivated by the fact that vehicles are often rear-loaded and, consequently, the load rearrangement required by a mixed service is difficult to implement in practice. The linehaul customers often have a higher service priority than the backhaul customers.

An example of a VRPB is that of the consumer goods supply chain, where the customers of a company are regarded as linehaul customers and the suppliers as backhaul customers.

Toth and Vigo [13] developed an integer linear programming model to solve the VRPB. The model they proposed is based on the reformulation of VRPB as an asymmetric problem. They presented three relaxations on their model: Firstly, a Transportation Problem is obtained when the capacity-cut constraints in their model are discarded. Secondly, a relaxation of the problem is obtained based on the projection of the feasible solution space, which requires the determination of degree-constrained shortest spanning trees, as well as the optimal solution of minimum cost flow problems. Thirdly, a Lagrangian lower bound is derived.

Toth and Vigo [13] then developed a lowest-first branch-and-bound algorithm based on their Lagrangian relaxation for finding the optimal solution to the asymmetrical VRPB. The branching rule used in the algorithm is an extension of the sub-tour elimination scheme.

### 2.1.9. The split delivery VRP (SDVRP) and Rollon-Rolloff VRP (RRVRP)

There is a general restriction in vehicle routing problems that customers may be visited once and therefore may be visited by only one vehicle. In the split delivery vehicle routing problem (SDVRP) [14] this restriction is relaxed and customers can be served more than once and by various vehicles.

The relaxation of the restriction is required when the sizes of the customer demands are larger than the vehicle capacity. The relaxation also results in lower costs in problems where the customer demands are almost as large as the vehicle capacity.

The aim of the SDVRP is to minimise the routing cost of the vehicles subject to the same constraints as the classic VRP, except for the relaxation that allows split deliveries. A simple way of solving the SDVRP involves splitting the customer demands into smaller demands and to treat each demand as a separate customer in the classic VRP. A savings-based algorithm can then be applied to solve the problem.

A vehicle routing problem that is related to the SDVRP and VRPPD is the Rollon-Rolloff vehicle routing problem (RRVRP). The RRVRP was defined by Bodin et al. [15] and involves vehicles moving trailers between customers and the depot. Full trailers are moved by the vehicles from the customers to the depot where they are emptied and returned to the customers. Because of the size and weight of the trailers, they can only be moved one at a time. The aim of the RRVRP is to minimise the total travel time of vehicles. Bodin et al. [15] proposed four heuristics for solving the problem.

### 2.1.10. The stochastic VRP (SVRP)

The stochastic vehicle routing problem (SVRP) [16] is a group of vehicle routing problems where certain components of the problem are random. The components of the vehicle routing problem that are often regarded as being stochastic include:

1. the presence or absence of customers,
2. the size of the customer demands,
3. the service times of the customers, and
4. the transit times of the vehicles.

Stochastic vehicle routing problems are usually solved by determining a solution before a specific outcome is known. The solution is then adapted after an outcome has been reached.

### 2.1.11. The open VRP (OVRP)

Sariklis and Powell [17] defined an interesting variation of the VRP, the open vehicle routing problem (OVRP). The distinguishing feature of this problem is that the vehicles do not have to return to the depot, or if they are required to do so, they return on the same path along which they left. The vehicle routes are thus not closed circuits but open paths.

The OVRP originates from a situation where a company does not own a vehicle fleet and therefore contracts their product distribution to an external carrier. The hired fleet is assigned to routes that originate from the company's depot, but the vehicles do not have to return to the depot after serving the customers.

The aim of the OVRP is to minimise the total vehicle operating and transit cost. The problem consists of finding vehicle routes that are subject to similar constraints as the CVRP:

1. Each vehicle route starts at the depot, $v_0$, and ends at one of the customers.
2. Each vertex $v_j \in V\backslash\{v_0\}$ is visited exactly once by exactly one vehicle route.
3. The sum of the demands of the vertices visited in a vehicle route may not exceed the vehicle capacity $Q$.

Sariklis and Powell [17] proposed a "cluster first, route second" heuristic for solving the OVRP.

## 2.2. VRP SOLUTION ALGORITHMS

This section contains a short discussion of the solution algorithms used to solve vehicle routing problems. The algorithms are divided into three categories: exact methods, heuristic methods and metaheuristic methods. Each category is broadly discussed in the following sub-sections. For a more detailed discussion on heuristics and metaheuristics the reader is referred to the surveys of Laporte et al. [18] and Cordeau et al. [19].

### 2.2.1. Exact methods

Lenstra and Rinnooy Kan [20] proved that the vehicle routing problem is *NP*-hard. *NP*-hard problems require immense computational effort to solve optimally as the size of the problems increase. Accordingly, exact methods are usually applied to smaller vehicle routing problems.

The models used to formulate the VRP and the algorithms that are used to solve it optimally are discussed in this section

(a) Formulations used for exact methods

Integer linear programming (ILP) is usually used when solving the VRP optimally. Three formulations that are used commonly are:

1. vehicle flow formulations,
2. commodity flow formulations and
3. set partitioning formulations.

(a.i) Vehicle flow formulations

In vehicle flow formulations integer variables are associated with each edge (or arc) of the graph which represent the number of times the edge (or arc) is traversed by a vehicle.

Vehicle flow formulations cannot handle situations where the objective function value of a solution is dependent on the overall customer sequence of a vehicle route or on the type of vehicle assigned to the vehicle route. The linear programming relaxations of these formulations can be weak when the additional operational constraints are tight.

Consequently, vehicle flow formulations are more suited for basic versions of the vehicle routing problem where the objective function value of a solution is expressed as the sum of the costs of the edges (or arcs) of the vehicle routes.

(a.ii) Commodity flow formulations

In commodity flow formulations additional integer variables are associated with edges (or arcs) which represent the quantity of goods (or commodities) that flow along the paths travelled by vehicles. Garvin et al. [21] first proposed this formulation when they considered an oil delivery problem. Commodity flow formulations are less often used to solve vehicle routing problems than vehicle flow formulations and set partitioning formulations.

(a.iii) Set partitioning formulations

Balinski and Quandt [22] were the first researches to solve the VRP using a set partitioning formulation. In set partitioning formulations, binary variables are associated with the different feasible vehicle routes. A set partitioning problem (SSP) is then used to find a collection of vehicle routes with a minimum total cost in which each customer is served exactly once and which satisfies other additional constraints imposed on the VRP.

The set partition formulation allows for very general route costs and the additional side constraints do not need to consider restrictions concerning the feasibility of a single route. Consequently, set partitioning formulations often have LP relaxations which are tighter than vehicle flow formulations and commodity flow formulations.

The disadvantage of this formulation is that an exponential number of binary variables are required to represent the different vehicle routes.

(b) Approaches used in finding optimal solutions

An overview is given in this section of three approaches that are frequently used to find optimal solutions to vehicle routing problems.

(b.i) Branch-and-bound algorithms

"Branch-and-bound" is, in its simplest, form a divide and conquer strategy, where a problem is broken up into sub-problems and the sub-problems are then evaluated and further divided. If a sub-problem results in a feasible solution it will not be divided further and the objective function value of the sub-problem forms a lower bound to the original maximisation problem.

If any sub-problem (in a maximisation problem) has an objective function value lower than the highest lower bound it will not be further evaluated since any feasible solution resulting from the sub-problem will be sub-optimal.

An optimal solution is found if a feasible solution can be found that is as high as an upper bound for the problem or if all branches have been fathomed. The fathoming of branches can be speeded up by finding high lower bounds so that a branch can quickly be discarded. Consequently, much effort is spent on finding bounding procedures for the vehicle routing problem.

Various relaxations of the VRP problem have been proposed, since the relaxation of a VRP problem forms a lower bound to the original problem. Two basic relaxations of the CVRP are:

1. the relaxation proposed by Laporte et al. [23] by reformulating the CVRP as Transportation Problem by dropping the capacity-cut-constraints and
2. the relaxation obtained by removing the out-degree constraints imposed on all the customers and weakening the capacity-cut-constraints so that only the connectivity of the solution is imposed.

Fischetti et al. [24] proposed two better relaxations for the CVRP, one based on a disjunction on infeasible arc subsets, and one based on a min-cost flow relaxation. These were combined according to the additive approach of Fishetti and Toth [25]. Lower bounds were also obtained by Fisher [26] and Miller [27] by means of Lagrangian relaxations. Hadjiconstantinou et al. [28] used a heuristic algorithm to solve the dual of the LP relaxation of the set partitioning formulation of the CVRP to obtain a lower bound.

Two relaxations that have been derived for the VRPTW is:

1. a network lower bound (obtained by relaxing the time window and capacity constraints) and
2. a linear programming lower bound (obtained by reformulating some of the binary conditions as inequalities).

Two branching strategies that are often used are the best-bound-first strategy (where branching is done on the node with the lowest lower bound value), and the depth-first strategy (where branching is done based on a last-in-first-out rule).

There are various ways to perform branching. Christofides et al. [29] proposed a branching scheme known as "branching on arcs" where branching is done to extend partial paths which start from the depot and finish at a given vertex. In this scheme, a branch at a node consists of using or not using an arc to extend the current path. Another branching scheme that is also frequently used is known as "branching on customers" and involves adding customers to the current sequence of customers in the partial path. Branching in this way is done by

selecting one of the arcs that join the end vertex of the sequence with a non-visited customer. (An extra node is added which represents the option of adding no further customer to the specific end vertex of the current sequence.)

(b.ii) Branch-and-cut algorithms

Integer linear programming problems with a small number of constraints are often solved by means of branch-and-bound algorithms. Branching is performed on the fractional variable values of the solution of the LP relaxation of the ILP.

When the number of constraints of an ILP is large, a similar technique to branch-and-bound" (called "branch-and-cut") is used to solve the ILP.

In branch-and-cut an LP relaxation with a subset of the constraints of the original ILP is solved. If all the constraints of the LP relaxation of the original ILP (with all the constraints) are satisfied, an optimal solution to LP relaxation of the ILP or an optimal solution to the ILP itself has been obtained. If some constraints of LP relaxation (with all the constraints) are violated a separation algorithm is used to return some violated constraints.

These constraints are then added to the LP relaxation (which contains a subset of the constraints) and it is solved again. This process is repeated until an optimal solution to the ILP or its relaxation is obtained. (If the ILP is solved optimally the algorithm terminates.)
If the solution to the LP relaxation (with all the constraints) is obtained, branching is done on one of the integer variables with a fractional value. In the one branch an upper bound is added for the variable and a lower bound for the second branch. The algorithm is then reapplied to each branch until an optimal solution for the ILP is returned.

In 1995 Augerat et al. [30] used the branch-and-cut algorithm to solve two instances of the CVRP containing 135 vertices optimally. These are two of the largest CVRP problems solved to proven optimality. More recently, Lysgaard [31] et al. presented a branch-and-cut algorithm that could solve some instances of VRP to optimality for the first time and they provided the best known upper bounds for various others.

(b.iii) Set Partitioning methods

A VRP which is formulated as a set covering problem is solved optimally when a minimum-cost set of feasible vehicle routes is selected such that every customer is included in one of the selected routes. This formulation results in an integer linear programming problem with a

constraint matrix, in which the columns represent feasible vehicle routes and the rows represent the customers.

The set covering problem is solved by relaxing the 0-1 constraint that a vehicle route is either in the optimal solution or not. This LP relaxation is solved using the column generation technique. The optimal LP relaxation solution is then used as a lower bound on the ILP. A branch-and-bound or branch-and-cut approach is then used to find a near optimal solution to the ILP using the vehicle routes obtained from the column generation. If an optimal solution is required, additional vehicle routes are generated in conjunction with a branch-and-price approach.

The column generation technique is used in the process described above to avoid enumerating all vehicle routes. This technique involves enumerating an initial set of vehicle routes (or columns) and solving the LP relaxation using these vehicle routes. The dual variables (or shadow prices) of the optimal LP solutions are then used to identify columns (or vehicle routes) that should be included in the initial set. If no column can be included to improve the objective function the optimal solution to the LP relaxation has been found; otherwise the additional columns are added and the process repeated.

Desrochers et al. [32] applied the set partitioning method to the VRPTW successfully and solved problems with 100 customers optimally. They used dynamic programming as part of their column generation step to solve the LP relaxation. The LP relaxation provided an excellent lower bound to the ILP, which they solved with a branch-and-bound algorithm. In 27 problems out of 87 problems they attempted, the lower bound provided by the LP relaxation was equal to the optimal value of the ILP.

### 2.2.2. Heuristic methods

Heuristic and metaheuristic methods usually explore the solution space partially and return the best solution found. These methods seldom return the optimal solution but they are computationally efficient algorithms.

Heuristics are flexible and can be used for a variety of complex problems. The heuristics that are frequently used for solving the VRP are discussed below.

(a) Sweep algorithms

The sweep algorithm was developed in the early 1970s and is a very simple method for solving planar instances of the vehicle routing problem. Laporte et al. [18] state that the algorithm was developed by Wren and Holliday [33], although it is often attributed to Gillett and Miller [34].

Vehicle routes are created by choosing a vehicle and sweeping a ray around the depot. Customers are included in the vehicle route as the ray sweeps over them. This continues until the vehicle capacity or maximum route length has been reached. Another vehicle is then chosen and the sweeping continued until all customers have been assigned to a vehicle route. Figure 2.2 illustrates this process.

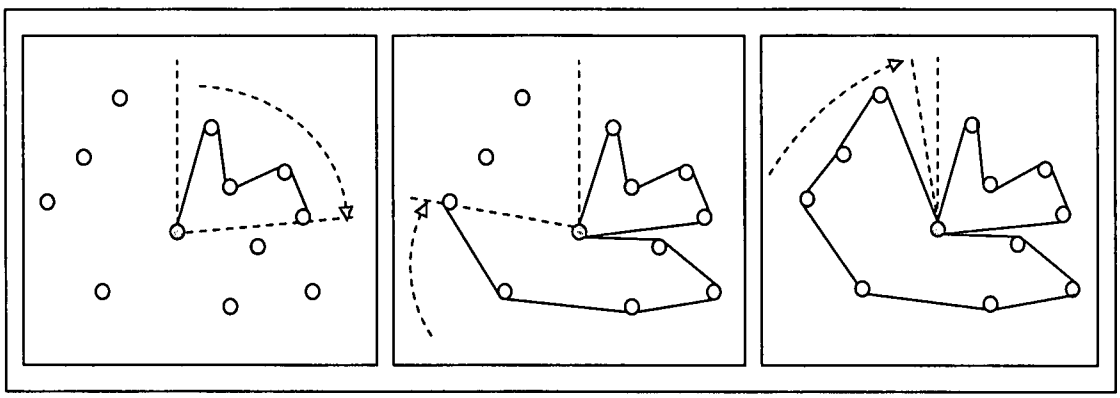


*Figure 2.2 Illustration of a sweep algorithm*

A travelling salesman problem (TSP) optimisation is then applied to each vehicle route. The travelling salesman problems can be solved with either exact algorithms or heuristic algorithms. A 3-opt step can also be applied to the initial solution obtained.

The sweep algorithm can be outlined as follows:

1. Rank the unassigned vertices according to the angle they make with an arbitrary axis originating from the depot.
2. Choose an unused vehicle $k$.
3. Assign the vertices to vehicle $k$ in the order they were ranked (in step 1) as long as the route length or vehicle capacity restriction is not violated. If some vertices could not be assigned to vehicle $k$, return to step 2.
4. Optimise each vehicle route (or circuit) by mapping it onto the travelling salesman problem. (Alternatively a 3-opt step can be applied to the vehicle routes as discussed later in this section.)

Step 4 is applied to the vehicle routes created in step 3, since the original route through the vertices in a vehicle route (created by the sweeping ray) is not necessary the shortest route. The shortest route through the vertices of a vehicle route can be found by solving a travelling salesman problem on the vehicle route.

Sweeping is often done in both the forward (counter-clockwise) and backward (clockwise) directions after which the best solution is chosen.

(b) Petal algorithms

Petal algorithms [35] are extensions of the sweep algorithm, where several routes (or petals) are created. A final selection of routes is then made by solving the following set partitioning problem:

Minimise $\sum_{k \in S} d_k x_k$

Subject to: $\sum_{k \in S} a_{ik} x_k = 1 \qquad i = 1, 2, \ldots, n$

$x_k = 0 \text{ or } 1, \quad k \in S$

where $S$ is the set of routes (or petals) generated, $x_k$ = 1 if petal $k$ belongs to the solution (otherwise $x_k$ = 0), $a_{ik}$ is a 0-1 parameter which is 1 if and only if vertex $i$ belongs to route $k$, and $d_k$ is the transit cost of petal $k$.

Figure 2.3 illustrates how a petal algorithm can be applied to a VRP problem.

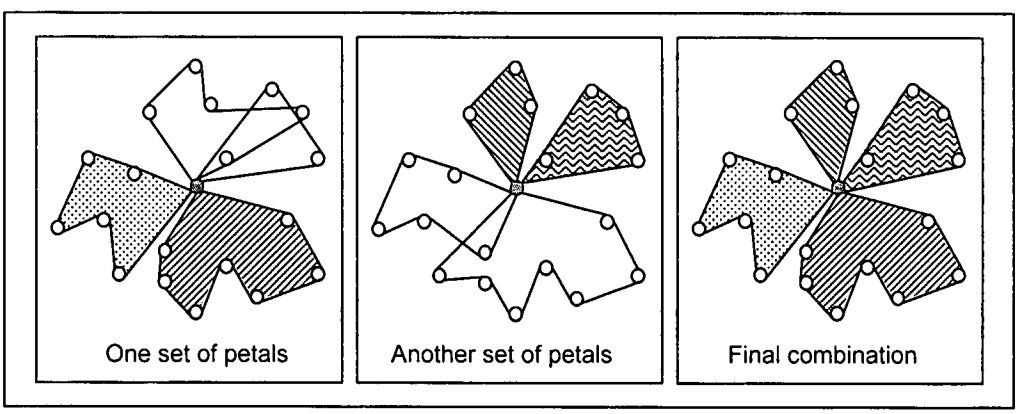



*Figure 2.3 Illustration of a petal algorithm*

(c) Savings-based algorithms

The Clarke and Wright [36] savings-based algorithm forms the basis for a variety of savings algorithms. The savings-based algorithms are used for solving problems where the fleet size is a decision variable.

The algorithm creates an initial set of routes after which it continually merges two routes, say $(v_0, \ldots, v_i, v_0)$ and $(v_0, v_j, \ldots, v_0)$, into a new route $(v_0, \ldots, v_i, v_j, \ldots, v_0)$ if a saving $s_{ij} > 0$ can be achieved by merging the two routes.

The saving $s_{ij}$ was originally defined by Clarke and Wright [36] as the cost (or distance) saved by a vehicle by extending its current route after visiting the last customer to include another vehicle route. Let $c_{xy}$ be the cost incurred by traversing arc $(v_x, v_y)$. The Clarke and Wright [36] saving is then defined as $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ when merging route $(v_0, \ldots, v_i, v_0)$ and route $(v_0, v_j, \ldots, v_0)$, into a new route $(v_0, \ldots, v_i, v_j, \ldots, v_0)$.

The savings-based algorithms can either be implemented in a sequential or parallel way. The parallel version is outlined as follows:

1. Find an initial solution by creating back and forth routes between the depot and all the customers.
2. Compute, store and rank the savings $s_{ij}$ for $i,j = 1, 2, \ldots, n$ and $i \neq j$.
3. Consider the highest ranked saving. Determine whether there are two vehicle routes – one ending with $(v_i, v_0)$ and another starting with $(v_0, v_j)$ – that can be merged to form a feasible solution. If a feasible solution can be achieved, merge these vehicle routes by adding $(v_i, v_j)$ and deleting $(v_i, v_0)$ and $(v_0, v_j)$. Delete the highest ranked saving.
4. Repeat step 3 until there are no more savings.

The sequential version differs from the parallel version in step 3. Instead of merging any two vehicle routes, the sequential version considers one vehicle route at a time, and merges other vehicle routes with it until no more mergings can be achieved. Another vehicle route is then chosen and the process repeated.

The diagram on the following page (Figure 2.4) illustrates how the parallel version is used to solve a VRP. Note that any two routes can be merged at any time.

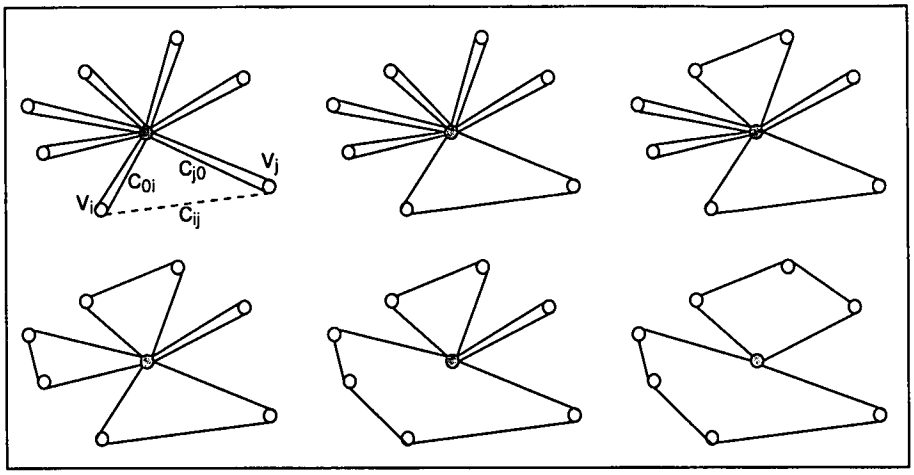



*Figure 2.4 Illustration of a parallel savings-based algorithm*

Figure 2.5 below illustrates how the sequential version is executed on a VRP. The greyed areas indicate which vehicle route is considered in each iteration.

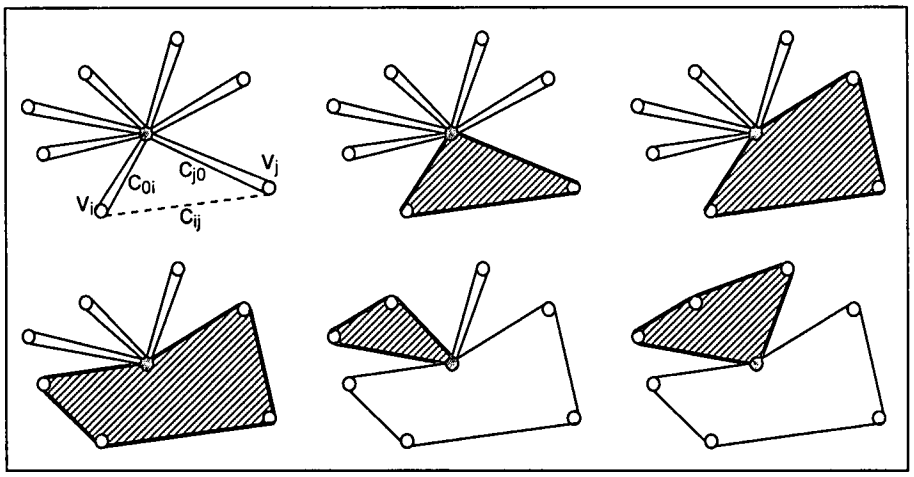



*Figure 2.5 Illustration of a sequential savings-based algorithm*

Renaud and Boctor [7] and Dullaert et al. [9] describe that Golden et al. [10] adapted the Clarke and Write [36] savings formula in order to solve the fleet size and mix VRP. The main adaptation made, the *combined saving*, includes the acquisition cost of vehicles in the savings formula. Two other adaptations, the *optimistic opportunity saving* and the *realistic opportunity saving*, were made to include the opportunity saving of unused vehicle capacity.

(d) Matching-based savings algorithms

Modifications made to the savings-based algorithms resulted in matching-based savings algorithms. The matching-based savings algorithm was originally proposed by Desrochers and Verhoog [37], but several modifications were later suggested by other researchers. (Examples of these can be found in Laporte et al. [18].)

In principle the saving $s_{ij}$ obtained by merging vehicle routes $p$ and $q$ containing vertex $i$ and vertex $j$ respectively, is replaced by:

$$s_{ij} = f(V_p) + f(V_q) - f(V_p \cup V_q)$$

where $V_p$ and $V_q$ are the vertex sets of route $p$ and route $q$, and $f(V_p)$ and $f(V_q)$ are the lengths of optimal travelling salesman problem solutions on $V_p$ and $V_q$.

The $s_{ij}$ values are then used as matching costs for a matching problem over the route sets. The vehicle routes corresponding to the optimal matching are then merged if the feasibility of the solution can be maintained.

The matching-based savings algorithms yield better solutions to the VRP than the simple savings-based algorithms, but at a higher computational effort. One way of reducing the computational effort required is to use approximated TSP solutions, rather than optimal ones.

(e) "Cluster first, route second" algorithms

Vehicle routing problems are sometimes solved by grouping (or clustering) customers together and finding a route through all the customers. If the clustering is done first, the approach is called "cluster first, route second"; otherwise if the routing is done before grouping the customers, the approach is referred to as "route first, cluster second".

The "cluster first, route second" algorithms are often used to solve multiple depot vehicle routing problems. The algorithms attempt to group the customers in clusters around a seed vertex (usually the depots in a MDVRP) and then find an optimal vehicle route through each cluster. It is important to note that the two phases of these algorithms are not independent – a bad clustering will result in a worse overall routing.

Figure 2.6 below illustrates how a "cluster first, route second" algorithm can be used to solve a MDVRP.

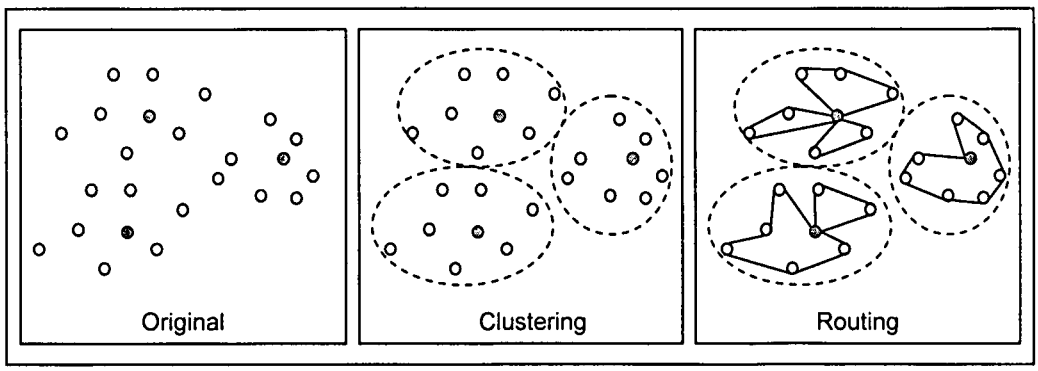



*Figure 2.6 Illustration of a "cluster first, route second" algorithm*

The most prominent "cluster first, route second" algorithm is the Fisher and Jaikumar [38] algorithm, which assumes a fixed fleet size. The Fisher and Jaikumar [38] algorithm, unlike other "cluster first, route second" algorithms, does not use a geometric method to assign customers to clusters. The Generalised Assignment Problem (GAP) is used instead.

The Fisher and Jaikumar Algorithm [38] can be outlined as follows:

1. Choose the vertices $j_k$ in $V$ that are to be used as seed points for each cluster $k$.
2. Calculate the cost $d_{ik}$ of assigning each customer $i$ to cluster $k$:

   $$d_{ij_k} = \min\{c_{0i} + c_{ij_k} + c_{j_k 0}; c_{0j_k} + c_{j_k i} + c_{i0}\} - (c_{0j_k} + c_{j_k 0})$$

   where $c_{xy}$ is the cost of traversing arc $(v_x, v_y)$.
3. Solve the GAP with costs $d_{ij}$, customer demands $q_i$ and vehicle capacity Q.
4. Solve the travelling salesman problem (approximately or exactly) for each cluster created by the GAP solution.

Although the Fisher and Jaikumar [38] algorithm is simple in principle, it is difficult to incorporate various constraints of vehicle routing problems into the algorithm. The capacity restrictions of the vehicles can easily be incorporated into the algorithm, but the route duration restrictions are much more difficult. Giosa et al. [5] developed six heuristics that can be used for the clustering part of these algorithms to allow time windows to be imposed on a multiple-depot VRP.

### (f) "Route first, cluster second" algorithms

The "route first, cluster second" algorithms (also referred to as giant tour algorithms) solve vehicle routing problems by initially constructing a single route through all the customers by ignoring the various constraints imposed on the VRP. This route construction can be done by solving a travelling salesman problem. After the single route has been constructed the route is decomposed into smaller feasible vehicle routes (or clusters).

Figure 2.7 illustrates how a "route first, cluster second" algorithm is used to solve a VRP (using single partitioning).

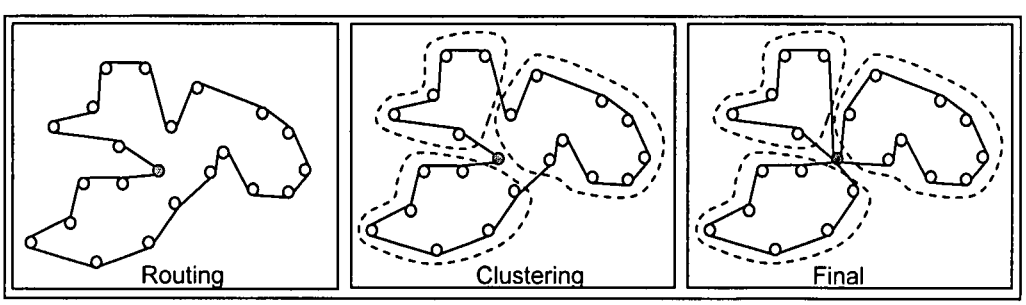



*Figure 2.7 Illustration of a "route first, cluster second" algorithm*

Renaud and Boctor [7] describe two versions of the algorithm, namely single partitioning and multiple partitioning. In single partitioning the depot is included in the initial route, where multiple partitioning excludes the depot. Multiple partitioning allows more flexibility, which results in a better clustering.

Few researchers use "route first, cluster second" algorithms as these algorithms cannot compete with other heuristics in simple vehicle routing problems. However, the algorithm is useful for solving fleet size and mix vehicle routing problems.

(g) Route improvement algorithms

The aim of route improvement algorithms is to improve a solution of a VRP by performing several operations on the solution. The operations can be performed on a single vehicle route or on several vehicle routes.

Lin's $\lambda$-opt [39] operator forms the basis for many route improvement algorithms. The $\lambda$-opt algorithm removes $\lambda$ edges from a solution and the remaining segments are reconnected in all possible combinations. If a reconnection is feasible and improves the solution, it is implemented. (Either the first or the best recombination is usually implemented.) Several modifications have been made to Lin's $\lambda$-opt [39] algorithm. Two well-known ones are the Or-opt [40] algorithm and the Lin and Kernighan algorithm [41].

Figure 2.8 illustrates how a 2-opt procedure is performed on a single vehicle route.

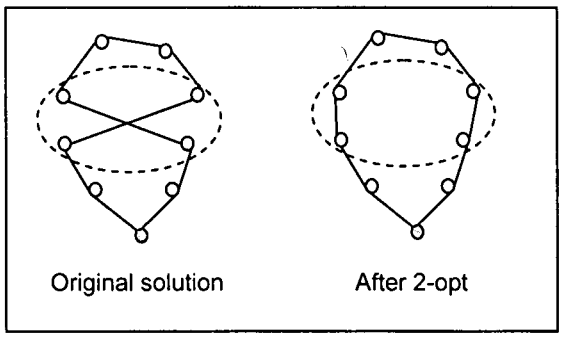



*Figure 2.8 Illustration of Lin's 2-opt algorithm*

Another well-known route improvement heuristic is Van Breedam's operators [42]. He defined four operations that can be performed on multiple routes:

1. String cross: Two strings of vertices are exchanged between two different routes by replacing two edges of two vehicle routes.
2. String exchange: Two strings (of at most $k$ vertices) are exchanged between two vehicle routes.

3. String relocation: A string (of at most $k$ vertices) is moved from one vehicle route to another.
4. String mix: The best move between string exchange and string relocation is chosen.

Figure 2.9 below illustrates how these operations can be performed on vehicle routes:

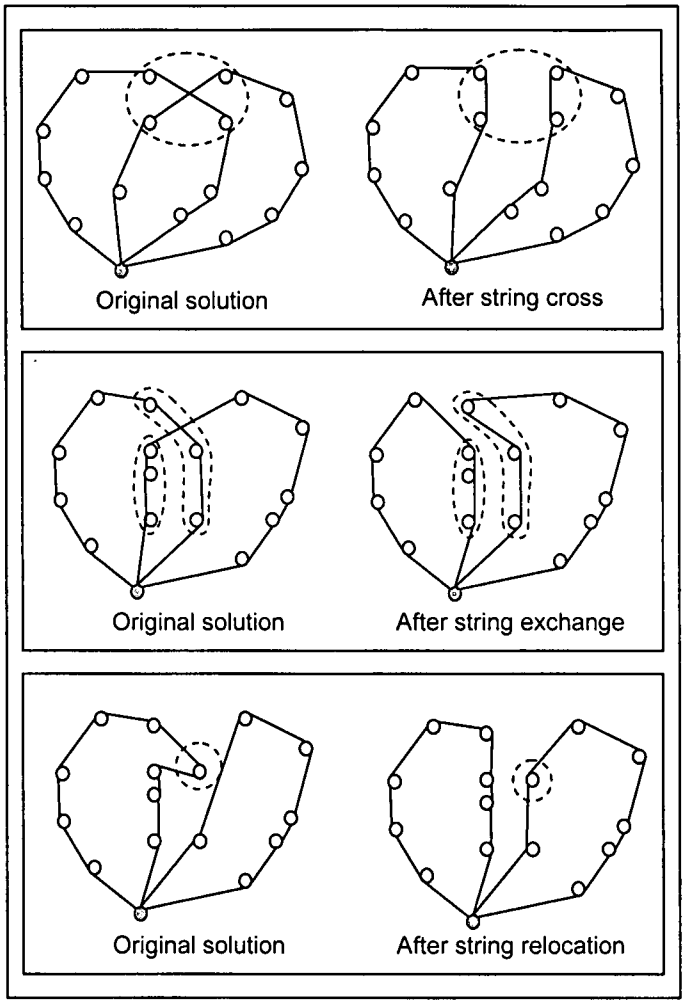



*Figure 2.9 Illustration of Van Breedam's operations*

Bachem et al. [43] introduced a new type of improvement heuristic for the VRP, called simulated trading. In simulated trading the vehicles "trade" customers assigned to their routes with one another, thereby simulating a typical process that takes place between truck drivers.

### 2.2.3. Metaheuristic methods

Metaheuristics are algorithms that explore the solution space to identify good solutions and often incorporate heuristic algorithms. Metaheuristics differ from heuristics as they allow intermediate deteriorating solutions (and sometimes infeasible solutions) in order to move from one local optimum to better local optima.

A variety of methods which fall into this category are discussed in the following paragraphs.

(a) Ant system algorithm

The ant system (AS) was originally developed by Marco Dorigo [44] in 1992 and has been applied to the Graph Colouring Problem, Job Shop Scheduling Problem and Quadratic Assignment Problem.

The ant system was first applied to the vehicle routing problem by Bullnheimer et al. [45] in 1997. They describe the underlying idea of the ant system in their 1999 paper [46] as follows:

> *The idea to imitate the behaviour of ant colonies for solving combinatorial optimization problems is based on observations made on real ants searching for food. It was found that real ants are able to communicate information concerning food sources via an aromatic essence called pheromone. While they move along, real ants lay down pheromone in a quantity that depends on the quality of the food source discovered. Other ants, observing the pheromone trail, are attracted to follow it. Thus, the path will be reinforced and will therefore attract more ants. Paths leading to rich, nearby food sources will be more frequented and consequently the corresponding pheromone trails will grow faster.*

When the ant system is applied to the vehicle routing problem, a simulated ant represents a vehicle. A route is constructed for each ant by successively choosing customers to visit until each customer has been visited. If the inclusion of a customer into an ant's route would lead to an infeasible solution (due to the vehicle capacity or total route length) the ant is returned to the depot.

An ant uses two aspects when deciding which customer to visit: Firstly, it considers how good *was* the choice of visiting the customer (given the ant's current position). This information is retrieved from the pheromone trail. Secondly, it evaluates how promising it *is* to visit the customer (given the ant's current position). The second aspect is determined by

means of a local heuristic and is referred to as the visibility. The probability of visiting each customer is determined by weighing these two aspects according to Dorigo and Gambardella's random-proportional rule [47].

After an ant has completed a route the pheromone levels on that route are updated according to the problem's objective function. When the pheromone is updated, a certain proportion of the old pheromone on the trail "evaporates" to avoid convergence to a local optimum and stagnation.

Bullnheimer et al. [45] suggested using a hybrid ant system by applying the 2-opt improvement heuristic to an ant's route before depositing pheromone on the trail. In early ant systems each ant deposited pheromone, but in the improved ant system of Bullnheimer et al. [46] only the ants with the best routes deposit pheromone. Each iteration in the improved ant system deposits extra pheromone on the arcs of the overall best solution obtained to further enhance the search algorithm.

Instead of a single colony of ants, multiple colonies can be used, each having its own unique pheromone marker. When multiple colonies are used the ant system is referred to as ant colony optimisation (ACO). Current research by Bell and McMullen [48] suggest that ant colony optimisation should be used when the number of vehicles in a VRP problem increases. It seems beneficial to represent each vehicle by a separate colony as this separates the most likely routes for the different vehicles.

The ant system algorithm is competitive with other metaheuristics and it is easy to implement the capacity and distance constraints of VRP. However, there does not seem to be an easy method to incorporate the wide variety of other constraints of practical problems in the ant system algorithm.

(b) Genetic algorithms

The genetic algorithm (GA) developed by Holland [49] is an adaptive heuristic search method that mimics the evolution process that takes place through natural selection. It was first applied to the VRP in the early 1990s.

In a genetic algorithm an initial solution population is created and a number of candidates selected. Many selection strategies exist, the simplest being ranking selection in which the

candidates are ranked according to their objective function values. The probability of selecting a candidate is then expressed as a function of its rank.

The selected candidates are then recombined to produce a next generation and the process is repeated. The recombination of candidates is achieved through a crossover procedure. In its simplest form, the crossover procedure is an exchange of one or more parts of the parent solutions to form the next generation. The exchanging of parts (of parents) is achieved by exchanging binary strings which represent the parameter values of the parents.

As in nature, a small probability exists that a mutation can take place when parent solutions are combined. The mutation procedure is an attempt to escape from local optimum values. An efficient genetic algorithm maintains a balance between genetic quality (obtained through evolution) and genetic diversity (created by mutation). The mutation probability is usually very low.

Significant features of the solution space of a problem must be represented as chromosomes, and a solution is uniquely identified through its genes in the chromosomes. A variety of methods have been used to represent the VRP in such a way that it can be solved by means of a GA. Initially, a good ordering of customers was sought with the GA, while the routing was done afterwards. This approach was then later used as part of a "cluster first, route second" heuristic. Another approach taken was to create child solutions by connecting two route segments from different parent solutions, or by replacing a route of the one parent with a route in the other parent. It was also suggested that child solutions could be created by disconnecting certain vertices from the parent solutions and reconnecting them by means of a route-construction heuristic.

Although these GA heuristics proved to be efficient algorithms for solving the VRPTW (with vehicle capacity constraints), they have for a long time not been able to compete with other metaheuristics in solving the VRP with capacity or distance constraints. Recently Baker and Ayechew [50] and Prins [51] have developed hybridised genetic algorithms which place the generic algorithm on a par with other metaheuristics used for solving vehicle routing problems.

The multiple-depot VRP and the VRP with backhauls have been solved with genetic algorithms, but most genetic algorithms developed so far are aimed at the VRPTW and CDVRP.

(c) Tabu search algorithms

Tabu search (TS) algorithms are currently revered as the most powerful metaheuristic for solving vehicle routing problems. These algorithms are often very complex as they are heuristics within heuristics, therefore only a short description is stated.

Laporte et al. [18] gives the following description of tabu search algorithms:

> *Tabu Search starts from an initial solution* $x_1$ *and moves at each iteration* t *from* $x_t$ *to its best neighbour* $x_{t+1}$, *until a stopping criterion is satisfied. If* f(x) *denotes the cost of* x, *then* $f(x_{t+1})$ *is not necessarily less than* $f(x_t)$. *To avoid cycling, solutions that were recently examined are forbidden, or tabu, for a number of iterations. To alleviate time and memory requirements, it is customary to record an attribute of tabu solutions rather than the solutions themselves.*

Although a solution may have been declared tabu, the tabu status may sometimes be overridden when a so-called *aspiration* criterion is met (e.g. when a tabu solution is the best solution obtained).

The reason the search is allowed to move from one solution to another solution, although the latter solution may be less profitable, is that it prevents the tabu search from becoming stationary in local optimum. Consequently a larger region of the solution space will be searched.

Important algorithms that belong to this algorithm category are the taburoute algorithm, Taillard's algorithm [52], Xu and Kelly's algorithm [53], the granular tabu search (by Toth and Vigo [54]) and the unified tabu search algorithm [55].

Rochat and Taillard [56] developed a tabu search algorithm hybridised with a genetic algorithm called the adaptive memory procedure (AMP). A population of good solutions is stored in a memory and continually updated by adding new high quality solutions and removing weaker ones. New solutions are formed by combining some routes of the solutions in the population using a tabu search.

(d) Simulated and deterministic annealing algorithms

Simulated annealing (SA) [57] algorithms are heuristic methods that are based on an analogy from the annealing process of a solid that gradually cools down in order to find a state of lowest energy.

A simulated annealing algorithm repeatedly changes an initial solution by making small alterations to the solution and either accepting or rejecting the alterations. If Δ is the change achieved in the objective function by the alteration and the change is favourable, the alteration is automatically accepted. If Δ is deemed unfavourable it can still be accepted with a certain probability $p_n$ (usually specified as $p_n$ = **Exp**(-Δ/$T_n$) where $T_n$ is a control parameter called the *temperature*). The acceptance of an unfavourable alteration allows the algorithm to escape local optimum values.

More formally we can say the algorithm randomly chooses a solution $x_{new}$ at iteration $n$ from the neighbourhood of the current solution $x_n$,

if Δ is favourable $x_{n+1} = x_{new}$

if Δ is not favourable $x_{n+1} = \begin{cases} x_{new} & \text{with probability } p_n \\ x_n & \text{with probability } (1 - p_n) \end{cases}$

An SA algorithm usually lowers the temperature from an initially large value, according to a *cooling schedule* or *cooling function*. A typical cooling schedule is to reduce the temperature after every $N$ iterations so that: $T_{n+1} = \alpha T_n$ where $0 < \alpha < 1$. The probability of accepting a worse solution is consequently reduced over time.

The algorithm stops when a certain criterion is reached. Three stop criteria that are commonly used are the following:

1. the objective function value has not decreased by at least $\beta_1$% for at least $k_1$ consecutive cycles of $N$ iterations;
2. the number of accepted moves has been less than $\beta_2$% for at least $k_2$ consecutive cycles of $N$ iterations;
3. and $k_3$ cycles of $N$ iterations have been executed.

Deterministic annealing uses the same concept as simulated annealing, but a deterministic rule is used for the acceptance of an alteration. Two deterministic annealing algorithm types are threshold acceptance algorithms and record-to-record travel algorithms.

Suppose $f(x)$ is the objective function value of solution $x$ of a minimisation problem, then the threshold acceptance algorithms state that $x_{new}$ is accepted if $f(x_{new}) - f(x_n) < T_h$, where $T_h$ is a control parameter called the threshold ($T_h > 0$). The threshold is reduced during the search process.

In record-to-record travel algorithms $x_{new}$ is accepted if $f(x_{new}) < \beta.f(x_n)$, where $\beta$ is a user-controlled parameter slightly larger than one. During the search $\beta$ can gradually be reduced to 1.

Tarantilis et al. [11] developed an effective threshold acceptance heuristic for solving the HFFVRP called backtracking adaptive threshold accepting or BATA. Instead of reducing the threshold in a monotonic way, it is allowed to be increased if the current solution seems to have stagnated. The threshold is thus allowed to backtrack on its previous values, creating an oscillation effect which changes the search strategy between intensification and diversification.

## 2.3. PRACTICAL IMPLEMENTATIONS OF VEHICLE ROUTING PROBLEMS

Two practical implementations of vehicle routing problems are discussed in this section.

### 2.3.1. A vehicle carrier problem

Agbegha et al. [58] consider a similar vehicle carrier problem to the problem considered in this project. The Auto-carrier problem (ACP) which Agbergha et al. [58] considered can be described as follows: Given a group of vehicles of various sizes destined for dealers where the number of dealers is less than or equal to the number of vehicles, find an assignment of the vehicles to the carrier slots in a configuration that will not violate the space restrictions of the carrier, and which will minimise the combined cost of the routing of carriers and unloading and reloading of vehicles.

They state that the following decisions need to be made with the objective of minimising the combined cost of routing and unloading and reloading:

1. the selection of a group of vehicles (and associated dealers) to be delivered.
2. the partitioning of the dealers into groups, where the dealers in each group are to be serviced by a carrier and the assignment of a carrier to each group.
3. the development of the route which the carrier of each group will follow.
4. the allocation of vehicles to carrier slots based on the carrier route. (This is to say the loading of vehicles.)

The loading aspect of the ACP is an important part of the problem, since it is infrequent that the vehicles of only one dealer are placed on a single carrier. Few dealers in the cases they considered order volumes to permit this, customer service considerations do permit full carrier loads to be accumulated before a carrier is dispatched, and the number of high-profile vehicles in the sales mix limits the number of acceptable loading patterns of a carrier. The cost of reloading vehicles on a carrier can also be significant. Since the loading aspect of the problem is so important they focussed their research mainly on the loading aspect of the problem and developed an algorithm to solve the loading problem optimally.

### 2.3.2. A feeding distribution problem

An unusual VRP problem is described by Ruiz et al. [59]. They considered a feeding distribution problem where an average of 70 customers (of a possible 700) must be supplied from 11 depots every day. Each vehicle in the fleet has a loading capacity ranging from 12 to 24 ton, which is divided into a number of watertight compartments.

A unique characteristic is that only one customer can be served from a compartment, since different customers require different products that may not be mixed. The number of customers that can be served by a vehicle depends on the number of compartments in the vehicle. Furthermore, the total weight of the load may not exceed the maximum load restriction of the vehicle, and not all vehicles can travel on all edges.

The problem Ruiz et al. [59] considered has a complicated cost structure, since penalties are paid for not fully exploiting the loading capacity of a vehicle. A decision support system was developed in which the VRP was solved by a two-stage exact approach. In the first stage all possible feasible routes are generated, whereafter an integer programming model selects the optimum routes from all the routes generated.

The following remark in their article is noteworthy:

> *We completely agree with Desrochers et al. [60] when they remarked that the application of the techniques* [that] *appeared in the literature required a great deal of knowledge and expertise because of the existing gap between the problems considered in the literature and the real problems found in practice... [Implementation] of all aforementioned techniques in plants and factories is not so straightforward. ... As we have seen, many of the techniques do not provide the flexibility and responsiveness that a real logistic environment needs. Many of the cited references deal with simplistic versions of the VRP, which do not take into account heterogeneous vehicle fleets or other optimization criteria different from total route length minimization...*

## 2.4. CONCLUSION

The problem addressed in this thesis has many aspects that are similar to the vehicle routing problems examined in the literature. The similarities can be categorised as follows:

*Similarities in the fleet:* The fleet in the problem addressed in this thesis is limited in size, has capacitated vehicles and is heterogeneous. These aspects are similar to those of the CVRP, m-VRPTW and FSMVRP. The vehicles are not required to return to a central depot, which is an aspect modelled by the open VRP (OVRP). The vehicle capacities are small compared with the customer demands and split deliveries are allowed, as in the SDVRP and RRVRP.

*Similarities in the customers:* Customers are allowed to send and receive goods as in the VRPB. At first glance, the VRP models with multiple depots look attractive in the light of the Autocarriers problem, since the manufacturers can possibly be modelled as depots, and the branches as customers receiving goods from these depots. Soft time windows are imposed on the goods to be delivered (and for that matter on the customer demands), therefore the VRP models with time windows seem very promising. The customer demands are stochastic, as was the case with the stochastic VRP (SVRP).

Although there are many aspects that are similar to the different VRP models, none of these models can adequately describe the Autocarriers problem. The fundamental differences are as follows:

*Customer characteristics:* In the problem examined in this project customers can be visited by a number of carriers. Split deliveries and split pickups are required since the customer demands can exceed the vehicle capacity and goods can be temporarily stored at the vertices. While many VRP models only allow for customers to be visited once, a customer can be visited more than once in the Autocarriers problem. Since the vertices can store vehicles temporarily, the vertices have both linehaul and backhaul operations.

*Commodities characteristics:* It is often assumed in the solution algorithms developed for VRP problems that the products transported can be modelled as a single commodity. This is certainly true for a wide variety of problems, but not in the case of Autocarriers. Autocarriers transports two commodities – vehicles that can be loaded onto high slots only, and vehicles that can fit onto any slot of a carrier. In the Autocarriers problem the carriers have complex loading rules due to the physical nature of the carriers.

*Fleet characteristics:* Most, although not all, VRP models assume a sufficiently large fleet. Usually the fleet is also assumed to be homogeneous. The models that do not make these assumptions often incorporate the acquisition cost of the vehicles into the objective function. In the problem addressed, the acquisition costs of vehicles are not considered. Furthermore, the vehicles are not based at certain locations to which they must return. In the open VRP, vehicles do not have to return to the depot and thus their routes do not form a closed circuit. The vehicles do, however, return on the same path or disappear from the system, which is not the case for the problem under consideration.

*Role of the depot:* The depot plays an important role in many VRP instances, since goods move through the depot. In the Autocarriers problem there is no single depot through which goods move. The different branches of Autocarriers can possibly be modelled as depots, but this would change the nature of the problem. The branches of Autocarriers in such a model would be both the customers and the depots.

There are two reasons why exact methods can probably not be used to solve the Autocarriers problem. Firstly, an exact method would probably be computationally inefficient for solving this problem. Although there are not many customers, there are many vehicles that can be used to serve the customers, and the customers can be visited by various vehicles at various times. Therefore there are a number of possible routes. Secondly, it would be difficult to formulate the problem in such a way that it can be solved with an exact method, since goods can be dropped off at or picked up from intermediate vertices. This results in the vertices having split deliveries and pickups, as well as changing demands.

Metaheuristic algorithms have been successfully applied to many VRP problems and can possibly be used to find solutions to the Autocarriers problem. Heuristic algorithms are less flexible and often developed for specific problems. Some of the principles used in the heuristic algorithms can possibly be incorporated in a solution algorithm for the Autocarriers problem.

## 2.5. CHAPTER SUMMARY

The most important VRP models and solution algorithms were described in this chapter. It was found that the current VRP models are inadequate to describe the Autocarriers problem, since there are unique characteristics in the Autocarriers problem that distinguish it from these models. The following chapter contains a more detailed description of the Autocarriers problem.

# Chapter 3 Generic Problem Model

In order to develop a scheduling tool as described in chapter 1, a generic problem was defined. The definition was then used to develop a model that could be implemented in a software environment. This chapter contains a detailed description of the problem definition, the development of a scheduling model and the underlying assumptions of the model.

## 3.1. DEFINING THE PROBLEM

One of the aims of this project was to define a generic problem that can be used to solve a variety of similar problems. The generic problem was defined using a set of entities, namely: containers, transporters, vertices, arcs and movements, which are explained in detail in this section.

Containers are cargo generated at specific vertices (referred to as release vertices) and have to be transported to other vertices (referred to as destination vertices). The containers are transported by carriers (i.e. transporters) from the respective release vertices to the destination vertices through a set of vertices and arcs (which represent the allowable moves between vertices). Each arc joins one vertex with one other vertex in a specific direction. The graph defined by the set of vertices and arcs is referred to as the transportation network and represents allowable positions of transporters and containers.

Although every vertex does not have to be joined directly to every other vertex, the transportation network must form a connected graph so that every vertex can be reached from every other vertex.

The relationship between these entities can be represented graphically by the following diagram (Figure 3.1):

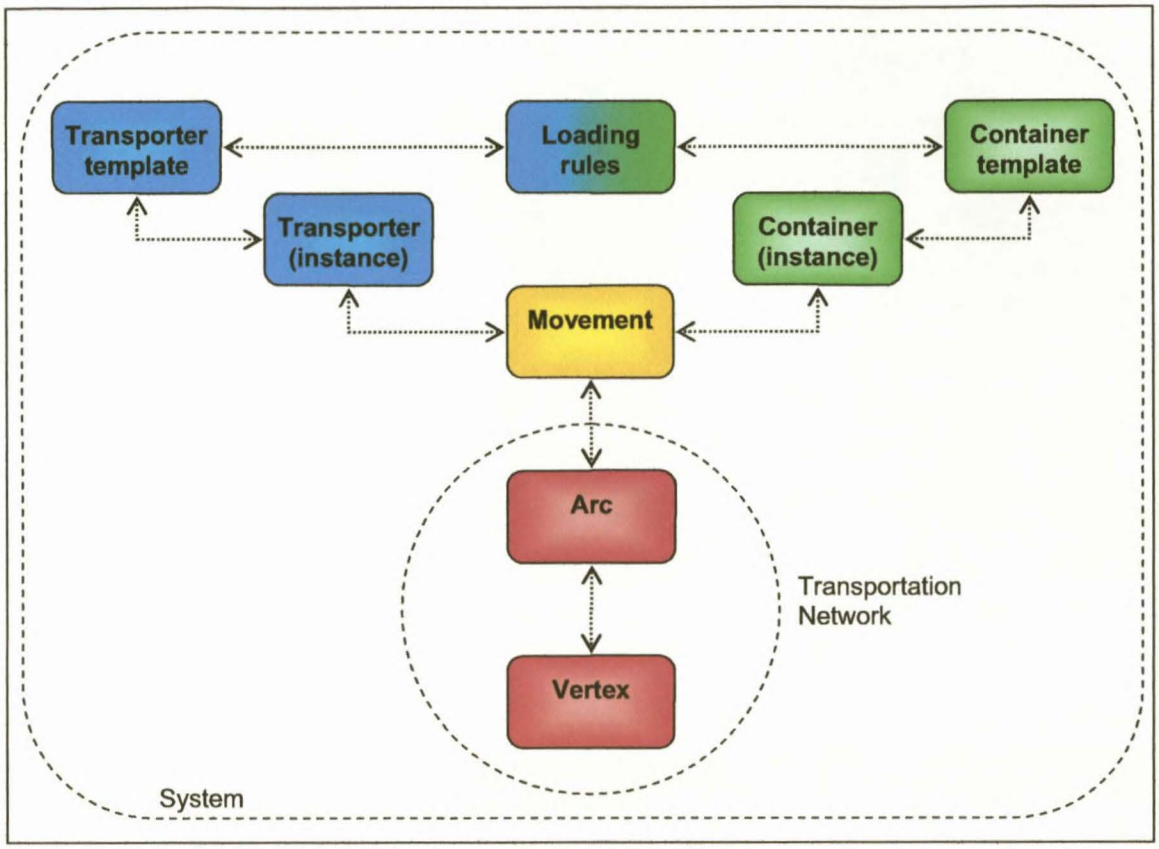



*Figure 3.1 Graphic representation of entity defintions*

The definition of each entity and its relationship to the other entities in the system are discussed in the following sub-sections.

### 3.1.1. Containers and Container Templates

A container is defined as a unit size of cargo and can represent a car (in the case of Autocarriers), a certain mass of material (e.g. a ton of wheat), or volume of material (e.g. 1 $m^3$ of oil).

Every container in the system must have the following attributes: a unique identifier so that a specific container can be addressed; a release vertex which indicates where the container will enter the system; a destination vertex representing the destination of a container; a release time stating when a container will be released into the system (i.e. when the container will be ready for pickup); a destination time which is the latest acceptable delivery time of the container; and lastly, a penalty rate. If a container is delivered after its destination time a penalty is paid at the appropriate rate, otherwise the penalty is discarded. The container can also optionally have an attribute to store the income received for transporting the container.

Since certain containers in the system display similar characteristics, they can be divided into categories or families called container templates. This makes it easier to describe and define relationships that include all containers that fall into the same category.

A container template may have attributes describing which containers fall under that category. It must have a unique identifier to distinguish itself from other container templates. The container templates are used to describe the cargo capacity restrictions of the transporters. This is further explained in the following two sub-sections.

### 3.1.2. Transporters and transporter templates

As explained earlier in this chapter, transporters are entities that move containers from their release vertices, through the transportation network, to the various destination vertices.

Each transporter has the following attributes: a unique identifier; a release vertex and release time which indicates when and where the transporter will enter the system, and lastly, a halt time representing the time when a transporter becomes unavailable for further transits.

In the same way that similar containers can be grouped together in container template families, the transporters can also be grouped into transporter template families, so that a group of similar transporters can be addressed together. Certain attributes of a transporter are derived from its corresponding transporter template. These attributes are the transporter's average travelling speed and its running cost (as a cost per kilometre). The transporter's cargo capacity restriction is also derived from its template as described in the following sub-section.

### 3.1.3. Loading rules

One of the unique characteristics of the generic problem addressed in this thesis is the limitations on the transporter's cargo capacity. Each transporter has a limited cargo capacity, which is a function of the number of each container type that is loaded onto it.

This problem characteristic can be explained using the Autocarriers problem as an example: Consider a carrier that has two decks: An upper deck with three slots which can hold any type of vehicle, and a lower deck with four slots which can only hold vehicles that are not higher than a certain height. The diagram below (Figure 3.2) shows an illustration of such a carrier.

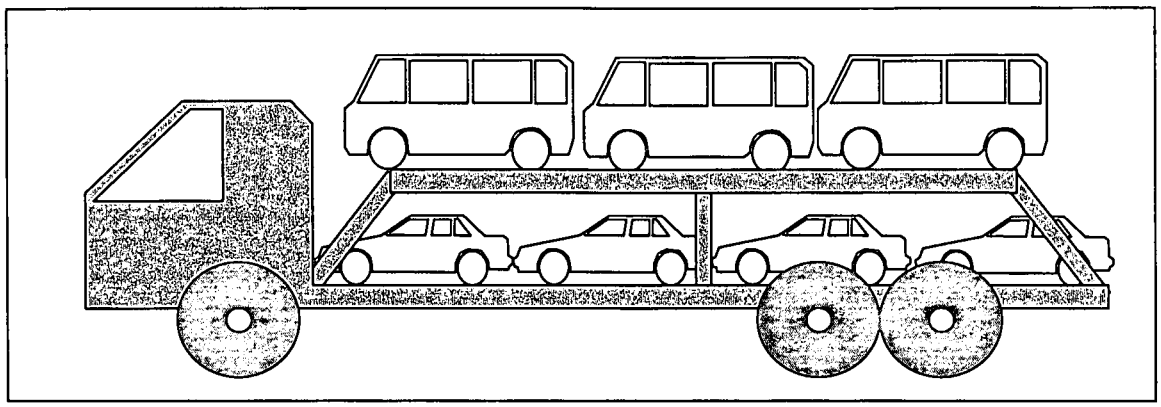


*Figure 3.2 Illustration of a carrier*

The cargo capacity restriction for the carrier holding three high vehicles and four low vehicles can be described by the following equations:

$n \leq 3$ and $m \leq 4$

$n, m \in N_0$

where $n$ is the number of high vehicles and $m$ is number of low vehicles.

The carrier can, however, also carry a different combination of vehicles. The following diagram (Figure 3.3) shows how the carrier can be loaded with low vehicles only.

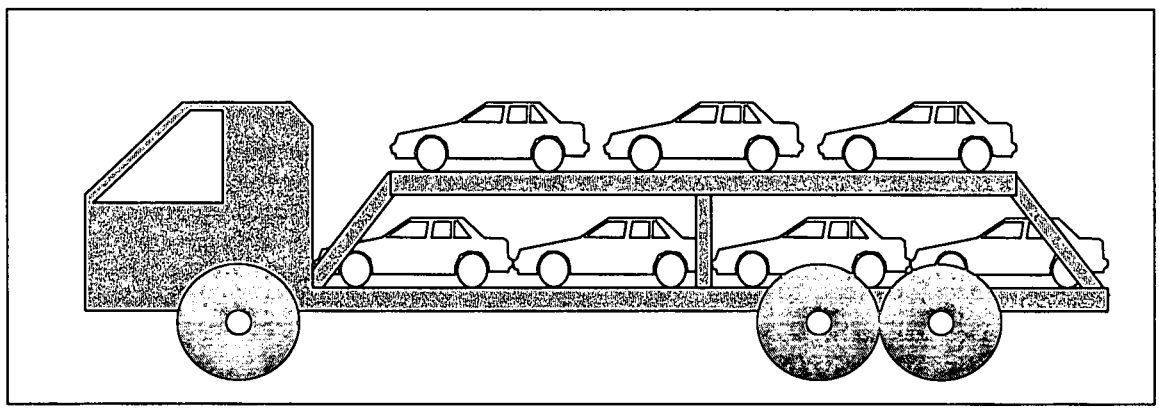


*Figure 3.3 Illustration of a carrier with a different load*

The cargo capacity restriction for the carrier holding this combination of vehicles can be described as follows:

$n = 0$ and $m \leq 7$

$n, m \in N_0$

where $n$ is the number of high vehicles and $m$ is number of low vehicles.

There are two more ways in which the carrier can be loaded: it can be loaded with one high vehicle and a maximum of six low vehicles, or with two high vehicles and a maximum of five low vehicles.

The cargo capacity restrictions of a transporter template can be expressed as a set of loading rules that represent the different loading configurations. In order to describe the full cargo capacity restriction of the carrier illustrated above, the following set of loading rules has to be used:

$(n = 0$ and $m \leq 7)$ or $(n \leq 1$ and $m \leq 6)$ or $(n \leq 2$ and $m \leq 5)$ or $(n \leq 3$ and $m \leq 4)$

$n, m \in N_0$

where $n$ is the number of high vehicles and $m$ is number of low vehicles.

The loading rules can be expressed in tabular form as follows (where $r_{xy}$ represents the maximum number of containers of type $x$ that can be loaded onto the carrier using loading rule $y$):

| | Loading rule 1 | Loading rule 2 | ... | Loading rule Y |
|---|---|---|---|---|
| Container template 1 | $r_{11}$ | $r_{12}$ | ... | $r_{1Y}$ |
| Container template 2 | $r_{21}$ | $r_{22}$ | ... | $r_{2Y}$ |
| ... | ... | ... | ... | |
| Container template X | $r_{X2}$ | $r_{X1}$ | ... | $r_{XY}$ |

*Table 3.1 Cargo capacity restriction in tabulated form*

As more container templates are defined, the number of loading rules (required to describe all the loading configurations) of a transporter template increases dramatically.

### 3.1.4. Vertices

In a system there are various locations where containers are released (i.e. generated) and delivered. There are also locations where transporters enter the system and there can be depots where containers are temporarily stored. All these locations are represented as a set of vertices.

Each vertex has a unique identifier and attributes that store the default values for the container penalty rate and container slack time of containers that are released from the vertex. The slack time is the time between a container's release time and destination time, which is the period in which a container can be transported without incurring a penalty. The vertices are connected by a set of arcs.

### 3.1.5. Arcs

An arc is a directional connection between two vertices. It specifies the allowable moves of transporters. The advantage of defining the connections between the vertices as directional connections is that the generic problem definition becomes more universal, since asymmetrical networks are included.

Problem instances where asymmetrical networks are present are problems that have the characteristic that the distances between vertices are dependent on the direction travelled. Two examples of this are the one-way streets in a city (in which case one distance is infinite) and the flight distance between cities. (The distance travelled by air between Johannesburg and Miami is longer that the distance between Miami and Johannesburg. In the former case an intermediate refuel is required, since an aeroplane cannot take off from Johannesburg with enough fuel due to the altitude.)

As stated earlier, the transportation network is defined as the collection of all the arcs and vertices in the system and represents all allowable positions of the transporters and containers.

### 3.1.6. Movements

A movement is defined as a single transporter travelling along one arc (of the transportation network). A movement therefore has an origin vertex, a destination vertex and a transporter associated with it.

Further attributes included are the start time, end time and cost of the movement. The end time and the cost of a movement can be calculated using the respective average speed and running cost rate of the transporter together with the distance traversed along the arc. The distance travelled by a transporter along an arc is also regarded as an attribute of the movement, so that the arc length can be changed without affecting past movements.

Since a transporter is associated with a movement, the template of the transporter is also associated with the movement and therefore the loading rules for the transporter template are associated as well. The indirect association of loading rules and movements enables the association of container templates and containers with movements. The containers associated (or assigned) to a movement are the containers carried by the transporter as it traverses the arc (at a certain time).

Certain conditions must be met in order for a movement entity to be a valid entity. These conditions can be summarised as follows:

1. Movements associated with a transporter must follow chronologically and logically so that the route of a transporter follows from consecutive vertices.
2. In the same way the movements associated with a container must follow chronologically and logically.
3. The number and types of containers associated with a movement may not violate the cargo capacity restrictions of the associated transporter.

Since the various entities of the generic problem have been defined, a scheduling model of the problem can be formulated. This model is described in the next section.

## 3.2. GENERIC SCHEDULING MODEL

Modelling the generic problem involved defining entities that can be used to describe the problem. The definitions of the entities were described in the previous section. This section contains a discussion of how the entities were used in the generic scheduling model. The model consists of an objective function and a solution space.

### 3.2.1. The objective function

The purpose of the objective function is to reflect the value of a specific scheduling solution. A good objective function therefore accurately reflects the business objective of a client. To keep the generic model universal, no specific objective function was defined. However, five important aspects of a solution were identified. These are deemed sufficient for most instances of the generic problem and a client may use any combination of these to form an objective function.

The five important aspects of a scheduling solution are:

- the on-time delivery of containers (referred to as the service level)
- the average container late time
- the total transporter running costs
- the total container penalty cost
- the efficient use of transporters.

Detailed definitions for these aspects are given on page 76 in Section 5.3.1. The aspects may result in conflicting objective functions and the client may have to value each aspect's importance. An example of a conflicting objective function is the minimisation of the transporter running costs and maximisation of the service level. A high service level requires the early delivery of containers and results in transporters travelling half empty. This, in turn, results in high transporter running costs.

### 3.2.2. Conditions of the solution space

The solution space is the set of all valid scheduling solutions and is a framework that defines the permissible associations among containers, transporters, arcs, vertices and movements. As mentioned on page 43 Sub-section 3.1.6, not all associations among these entities necessarily result in a valid solution.

The solution space is the set of solutions which comply with a set of conditions. The conditions ensure that valid movement entities are created and that valid associations will be

formed among all entities in the system. The conditions of the solution space are discussed in the following paragraphs as they apply to each entity of the system.

(a) Container conditions

1. A container must have an origin (i.e. a release) vertex and a destination vertex.
2. A container is allowed to be transported in the system by transporters only after it is released.
3. A container may be moved directly or indirectly (i.e. through other vertices) from its release vertex to its destination vertex. A container may be associated with more than one transporter if it is moved indirectly from its origin to its destination.
4. The consecutive transportations of a container must be allowed by the transportation network so that a container is not moved from one vertex to another if no arc exists between the vertices. This condition is automatically met if the conditions for the transporter moves are met.
5. A container is not allowed to leave a vertex before it has arrived or has been released at the vertex. This ensures that the transportation of the container follow coherently and chronologically.
6. Once a container has reached its destination, it is halted and may no longer be moved around in the system.
7. A container may not be associated with a movement, and therefore with a transporter, in a way that violates the cargo capacity restrictions of the transporter.
8. An optional condition that can be imposed is that a container may not be moved through the same vertex more than once. This condition does not affect the feasibility of a solution, but hinders a container from being moved cyclically which affect the optimality of a solution.

(b) Transporter conditions

1. A transporter enters the system by being released at a vertex at a certain point in time. A transporter is therefore not allowed to be released between two vertices or at any other location that is not defined in the transportation network.
2. A transporter must move coherently. A transporter cannot depart from a vertex before it has arrived and unloaded its cargo, and it cannot depart from a vertex at which it has not arrived (except in the case where a transporter has been released at the vertex).
3. A transporter may not be associated with containers in a way that will violate its cargo capacity restriction. This is automatically satisfied by the container conditions.
4. A transporter may or may not be halted in the system. If a transporter is associated with a movement that ends after its halt time, the transporter will only be removed once the

movement is completed. A transporter may not be associated with movements while it is halted.

5. A transporter that has been halted may be released again at the last vertex at which it arrived. This procedure makes transporter maintenance possible.

(c) Arc conditions

1. An arc must have an origin vertex and a destination vertex and only allows traffic in one direction. If two-way traffic between two vertices is required, two arcs in opposite directions must be defined to enable two-way flow.
2. There can only be one arc between two vertices in a certain direction.
3. At any point in time, an arc has the same length for all transporters.
4. Arcs may be added and removed from the system. The length of an arc may also be changed. If the length of an arc is changed while movements are associated with it, the transporters' arrival times and costs will not be affected. The arrival time of the containers associated with the movements will also stay unaffected. (The length of an arc may be changed when the distance between two vertices changes, for example, when a new road between two cities is built or an existing road becomes unavailable due to road maintenance.)

(d) Vertex conditions

1. Every vertex must be reachable from every other vertex, although the vertices do not need to be connected directly. If this condition is not met a container may not be able to reach its destination.
2. Vertices may be added and removed from a system. A vertex may not be removed from the system while there are containers destined for it.
3. Every vertex in the system (including the release vertices) must be able to function as a depot for containers so that any container may be stored temporarily at the vertex. This allows a container to move through the system by means of various movements and transporters.
4. The depot facilities at the vertices have an unlimited capacity so that there is no restriction on the maximum number of containers that can be temporarily stored at a vertex.

The result of these conditions is a guaranteed valid solution in which containers can flow from their origin to their destination by utilising one or more transporter movements of the various transporters travelling along the transportation network.

### 3.2.3. Modelling the solution space

One can use the conditions of the solution space, together with entity definitions, to model the generic problem definition as a graph theory problem. A series of diagrams are used to explain the modelling process.

The first step is to regard the transportation network as a graph. The diagram below (Figure 3.4) shows the geographical layout of a transportation network.

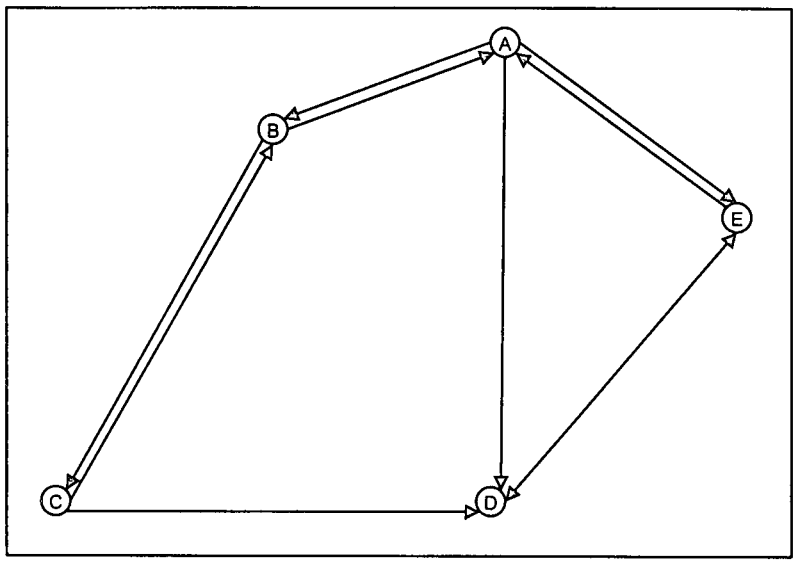



*Figure 3.4 Transportation network mapped on a graph*

The transporters are allowed to move around in the graph. As they move from one vertex to another, they create routes (i.e. movement entities). These movements are formed along the arcs of the graph obtained from the transportation network. The movements can also be represented as a set of arcs of a new graph referred to as the *movements graph*. (These arcs should not be confused with the arcs of the transportation network.) The following diagram (Figure 3.5) illustrates how two transporters have moved along the arcs of the transportation network to form five movements that are represented by the blue arcs (Transporter 1) and red arcs (Transporter 2). The diagram shows the movements graph imposed on the graph representing the transportation network.

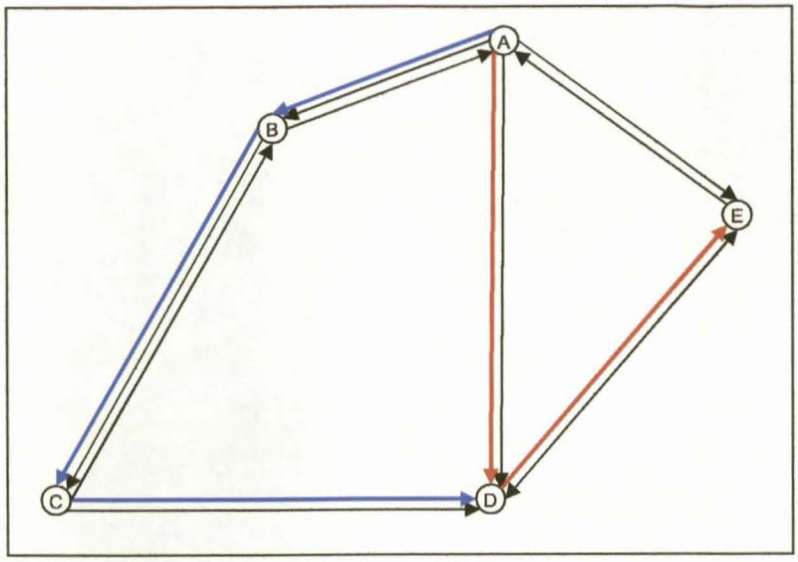



*Figure 3.5 Movements formed by two transporters*

Containers can now be associated with the movements graph to depict the transportation of the containers to and from the various vertices. Figure 3.6 below shows how containers (represented by the squares) can be moved along the movements graphs formed by the two transporters.

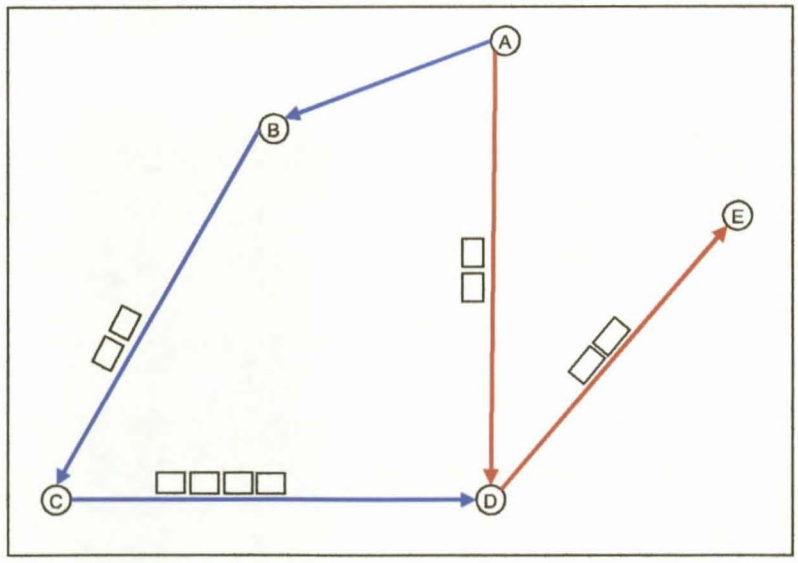



*Figure 3.6 Containers associated with movements*

Suppose a container has to be sent from vertex B to vertex E. One can examine the movements graph and find a path from B to E and associate the container with that path. The next diagram (Figure 3.7) shows how this container (represented by the grey square) is sent from vertex B to vertex E.

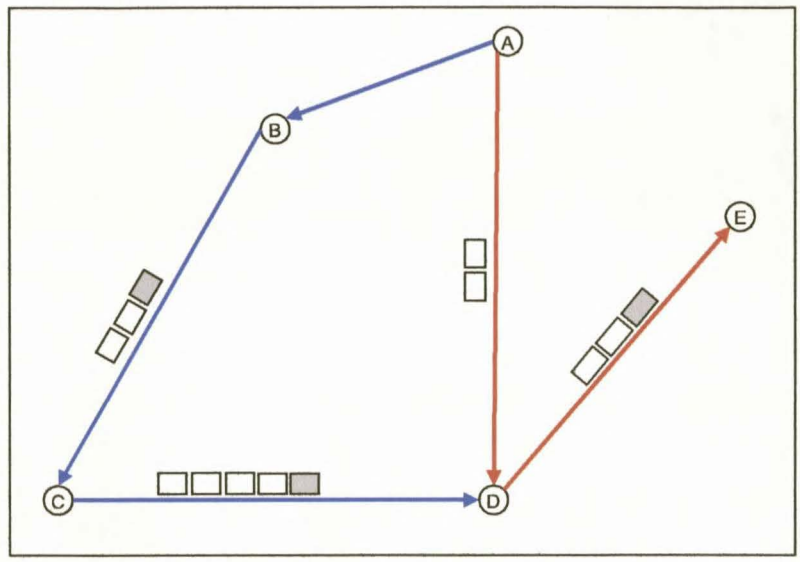



*Figure 3.7 Moving a single container in the system*

The diagrams above are useful in explaining the spatial restrictions on the transportation of containers and transporters, but they fail to show that these transportations form a time-dependent process. An idea from Davies [61] was used to describe the time dependency of the movements. The time dependency can be illustrated by the following diagram (Figure 3.8):

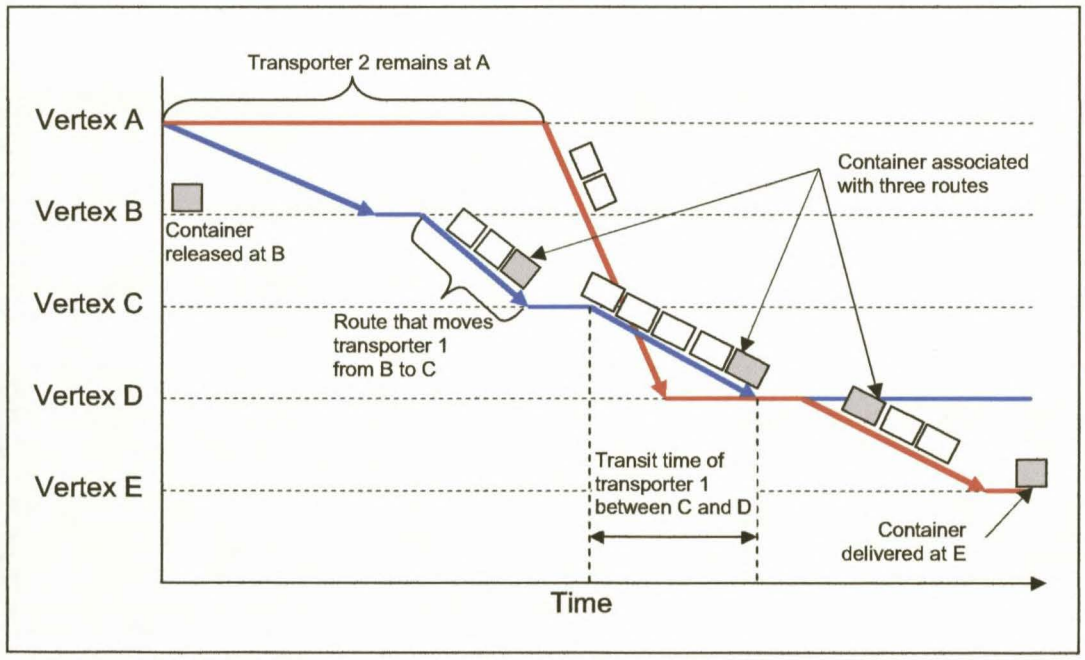



*Figure 3.8 Time representation of container movements*

In the figure above (Figure 3.8) the red and blue arcs indicate the moves of two transporters as illustrated in Figure 3.7. This representation illustrates that a container must arrive at a

vertex before it can depart from that vertex. Similarly a transporter cannot depart from a vertex before it arrives at the vertex and is reloaded with the next set of containers.

The time dependency illustration fails to illustrate the physical constraints on the transporters, that is to say, it does not show that a transporter cannot move from B to D as depicted by Figure 3.5. Therefore both representations are needed to describe a valid movements graph, which forms part of a scheduling solution.

A scheduling solution is a movements graph that is coherent within the time and spatial restrictions, with containers associated in a way that the containers are transported from their origins to their destinations without violating the transporters' capacity restriction.

An optimal solution can thus be defined as a scheduling solution that contains a valid movements graph with containers associated with it, which optimises the objective function. A scheduling algorithm will therefore have to generate the movements graph and assign containers to that graph.

The model developed for solving the generic problem was discussed in this section. This model has certain underlying assumptions, which are discussed in the next section.

## 3.3. ASSUMPTIONS OF THE GENERIC SCHEDULING MODEL

Some of the assumptions that were made in the development of the model were mentioned in the previous sections. It is important to assess the assumptions of a model before using it to solve real-world problems. It is for this reason that all the assumptions are consolidated in this section.

### 3.3.1. Assumptions concerning transporters

The model allows for a dynamic transporter fleet size, with various types of transporters, but it assumes that the fleet size is finite. The model is not developed for the case where there is an infinite number of transporters. It is further assumed that the transporter fleet size is large enough to move all the containers within a certain time frame. If containers are released too rapidly it may flood the transporters' ability to move all the containers to their destinations in a required scheduling period.

It is difficult to relate the maximum release rate of containers to the minimum number of transporters required, since there are many factors that influence this ratio. These factors include the number of vertices, the number of arcs and their lengths, the transporters' cargo capacity restrictions, the transporters' speed, the delay time of the transporter, and the algorithm used for solving the problem. (Some algorithms may make better use of the transporters than others and, consequently, they will require fewer transporters.)

The model assumes that the transit time of the transporters is only a function of the distance travelled. Transporters travelling with or without a load travel at the same speed. Similarly the uphill and downhill transit speeds of a transporter are the same. It is also assumed that there is no variation in the travelling speed of the transporter.

Once a transporter has reached a vertex, it has a certain delay time that has to elapse before it may leave the vertex. The delay time is the time it takes to unload, reload and refuel a transporter when necessary. It is assumed that the delay time is the same for all vertices and transporters and that it is independent of the load of the transporter and the time of day.

If there are big variations in the actual (i.e. the real world) transporters' speed, the variations can be compensated for by enlarging the delay time. If the transporter arrives late at a destination, some (if not all) of the late time will be absorbed by the larger delay time. Making the delay time too large may result in inefficient solutions, since the transporter's availability is reduced.

As stated earlier, the transporters have cargo capacity restrictions. A transporter may be relieved from the restriction by setting the maximum number of containers (of each type) in a loading rule to an appropriately large number. The model does not accommodate transporters which have contradictory loading rules. An example of contradictory loading rules can be stated as follows:

Rule 1: $n \leq 3$ and $m \leq 4$
Rule 2: $n \leq 3$ and $m \leq 3$
$n, m \in N_0$
where $n$ is the number of type 1 containers and $m$ is number of type 2 containers.

In the example above, rule 2 is contained within rule 1. Loading the transporter with three type 1 containers and four type 2 containers will result in rule 2 being violated, while rule 1 states that such a loading pattern is acceptable.

### 3.3.2. Assumptions concerning containers

A container is defined as the unit size cargo. That implies that the containers represent whole entities and therefore the model does not allow fractional units of cargo. Fractional units of cargo should be rounded up to the closest full unit so that it can be represented in the model. A further consequence of this assumption is that transporters can only carry an integer number of container entities.

A container is defined as an entity having an origin and a destination. A container's origin and destination must be represented by vertices in the model. The model does not allow containers to enter or leave the system at any other place than a vertex.

A container remains in the system while it has not reached its destination; once the destination has been reached, the container is halted in the system. A consequence of this is that the origin and destination vertices of a container cannot be the same vertex. Accordingly, a container that is to be transported from and to the same location cannot be represented in the model.

### 3.3.3. Assumptions concerning the transportation network

The transportation network is the set of vertices and set of arcs connecting the vertices. In order for a transportation network to be useful every vertex must be reachable from every other vertex. If a vertex is not reachable, a transporter will not be able to move to or from a vertex.

As discussed in the previous section, vertices function as release points, delivery points and depots for the containers. It is assumed that every vertex can function as a depot facility for the containers and that it has an unlimited storage capacity. This assumption requires careful consideration when describing a problem with the generic problem model.

An important assumption made about the arcs is that *every* transporter can travel on *every* arc. This may not be the case for all real-world problems, for example where containers are transported on land and sea. If a transporter represents a truck, the transporter will not be able to travel along an arc which represents a sea route. Similarly a transporter representing a ship will not be able to travel on arcs representing roads.

It is also assumed that an arc has the same length for every transporter. This assumption will not necessarily hold for a problem where two cities can be reached by various means. The distance between them may depend on the type of transporter involved.

### 3.3.4. Other assumptions

The model assumes that transporters' operators (i.e. drivers; pilots, etc.) do not have to be incorporated in the solution. This assumption simplifies the problem since a whole new set of entities does not have to be included in the problem. This assumption makes the generic problem more universal as it can be applied to a wide variety of problems, but it is a less realistic representation of a specific problem instance.

There are various restrictions that have to be taken into account if operators are to be included in the problem. These restrictions include labour laws, operator availability and the ability of the operators to operate the various transporters. The operator restrictions may differ significantly between the various real-world problems and it may be impossible to define a generic problem that incorporates all the required operator restrictions of all real-world problems and still keep the problem definition universal.

Twenty-four hour operation is also assumed for the generic model developed. This enables containers to be released at any time of the day. It also enables transporters to arrive or leave any time of the day. This assumption may or may not hold for a specific instance and thus requires careful consideration.

## 3.4. CHAPTER SUMMARY

The developed scheduling model was described in this chapter. The generic problem is defined in terms of the following entities: containers, transporters, vertices, arcs, movements and loading rules.

The entity definitions were used to develop a model that can be used to represent the instances of the generic problem. The model consists of an objective function and a solution space.

Certain assumptions were made about the nature of the problem and the model developed and these assumptions were discussed.

# Chapter 4 Conceptual Design of Software Tool

One of the aims of the project was to develop a software tool that can house the model discussed in the previous chapter. The software tool therefore enables the definition, solving and evaluation of a problem instance. The software tool also allows the simulation of various configurations of a problem instance and analysis of the input and output data of a simulation. This chapter contains a discussion of the conceptual design of the software tool for each of the main structures that make up the tool.

The software tool can be divided into the following structures:

1. The Problem Definition Structure
2. The Solution Development Structure
3. The Solution Evaluation Structure
4. The Scenario Simulation Structure
5. The Statistical Analysis Structure
6. The Database Structure

The first four structures are program structures that enable the creation of an instance of the generic problem and the development of solutions for that problem. The solutions can then be evaluated according to the desired objective function. Simulations can also be executed to examine various configurations of the problem instance as well as to determine the best automatic scheduling algorithm for a configuration.

In order to develop a simulation model that represents aspects of the real world, it may be useful to use the distributions found in historical data. The statistical analysis structure can be used to examine historical data as well as to make a deeper analysis of the simulation output.

The sixth structure mentioned is the database structure, which is used to store information about a specific instance of the generic problem. Since a specific instance of the generic problem is stored in a database structure, the instance can be duplicated, changed and evaluated through the same program structures.

The various structures and their interaction can be represented graphically as shown in the following diagram (Figure 4.1):

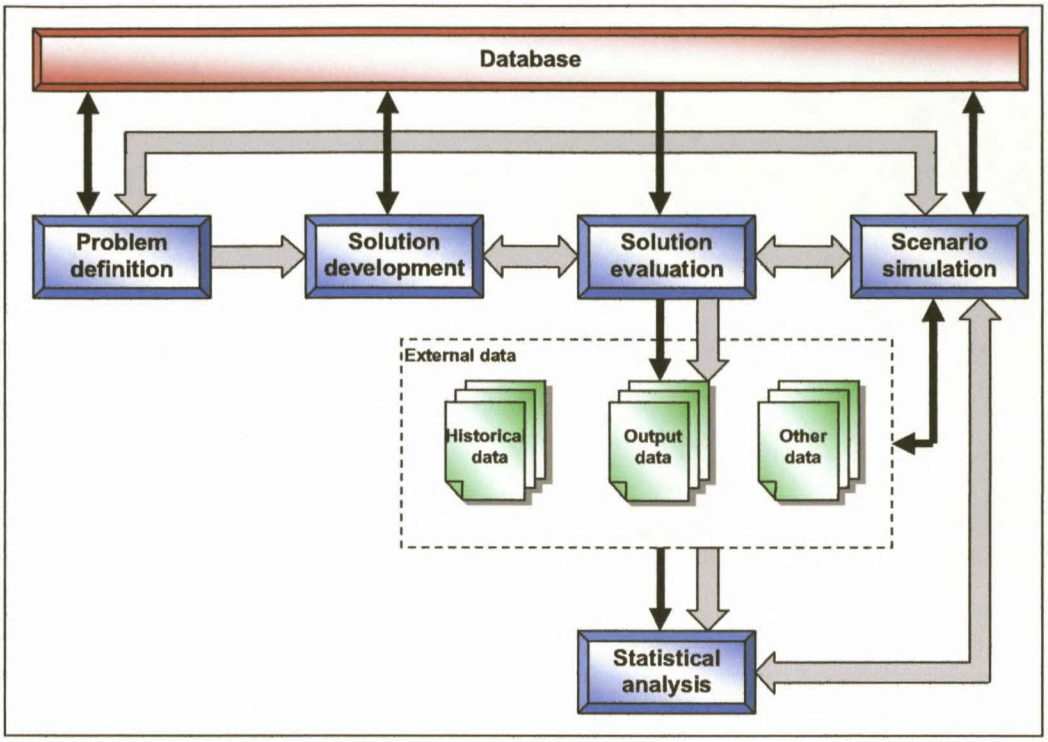



*Figure 4.1 Program and data structure with data and logical Flow*

In the diagram above, the blue squares represent the various program structures, while the red square represents the database structure. The black arrows represent the actual data flow between the various program structures, the database and external data. The grey arrows represent the logical flow of information.

The flow of information starts with the problem definition. An instance of the problem can be defined by means of the problem definition structure. The problem definition is used to develop a solution to the problem. This solution can then be evaluated by means of the solution evaluation structure. If the solution is not acceptable, other solutions can be generated and evaluated until a suitable solution is found. The solution evaluation structure allows for the export of data to text files. These files can then be examined statistically by the statistical analysis structure.

Once a problem has been defined, it is possible to simulate various scenarios by means of the scenario simulation structure. The output of each solution developed by means of a simulation can also be evaluated using the solution evaluation structure and the statistical analysis structure. The scenario simulation structure can make use of external data. The rest of this chapter provides a more detailed discussion of each structure of the software tool.

## 4.1. THE PROBLEM DEFINITION STRUCTURE

The purpose of the problem definition structure is to assist the user in defining a problem that is in accordance with the generic problem described earlier. The problem definition structure functions as a graphic user interface that aids the user in defining the following entities: transporter templates, transporters, containers, container templates and the transportation network. Since the transporter templates and container templates are defined in this structure, it must also assist the user in defining valid loading rules between these templates.

Five program sub-structures are used to allow for the creation of a problem definition. The sub-structures are: transporter template definition, transporter definition, container template definition, container definition and transportation network definition.

The following diagram (Figure 4.2) illustrates an entity breakdown of the problem definition structure:

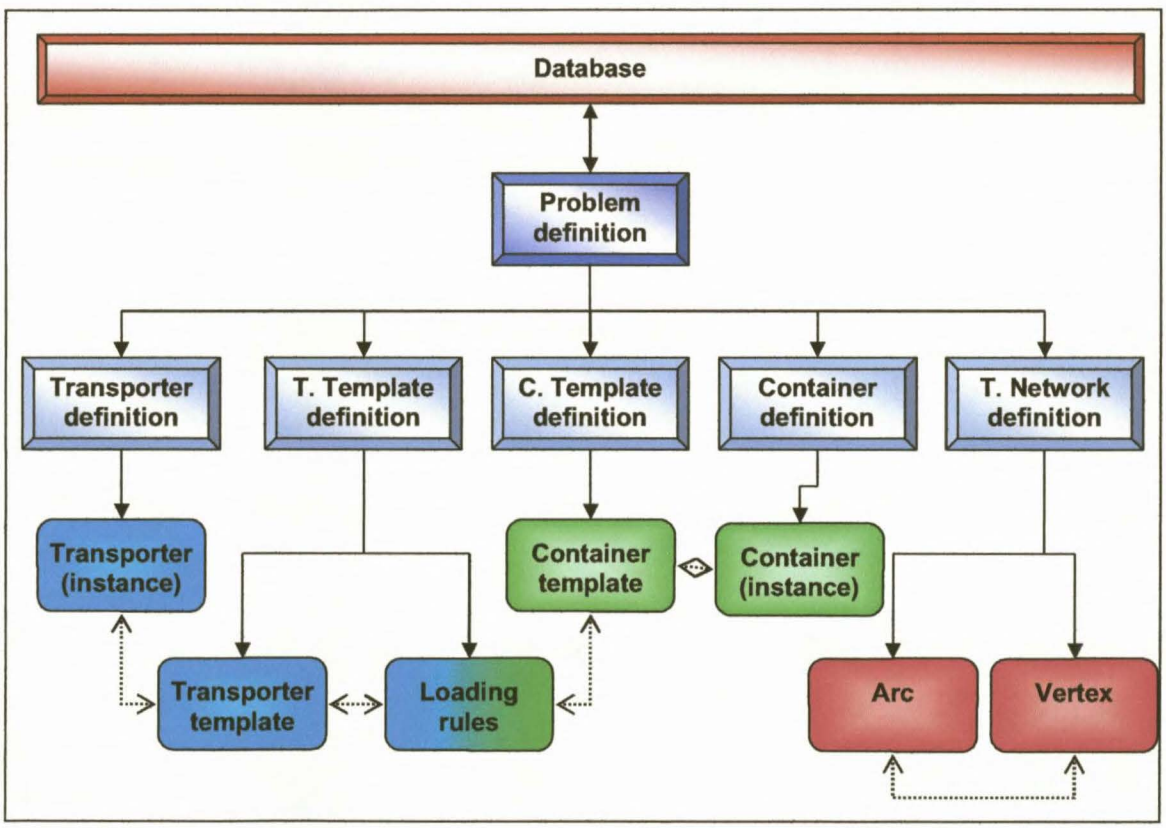



*Figure 4.2 Breakdown of problem definition structure*

As shown in the diagram above, the problem definition is stored in a database. This enables the other program structures to access the problem definition.

## 4.2. THE SOLUTION DEVELOPMENT STRUCTURE

The purpose of the solution development structure is to create a valid solution to the problem defined by the problem definition structure. A solution is generated by creating movements and assigning containers to the movements as described on page 48 Section 3.2.3. This structure ensures that a solution complies with all the conditions set for a valid solution space.

There are two program sub-structures that can be used to generate a solution. Firstly, a manual scheduling structure allows a user to create a solution. This structure makes it easier for a user to create a solution (compared with solving the problem by hand), since the program will ensure that all the conditions for a valid solution are met. Secondly, there is an automatic scheduling structure which automatically generates a solution according to a chosen heuristic.

The diagram below (Figure 4.3) illustrates an entity and structure breakdown of the solution development structure:

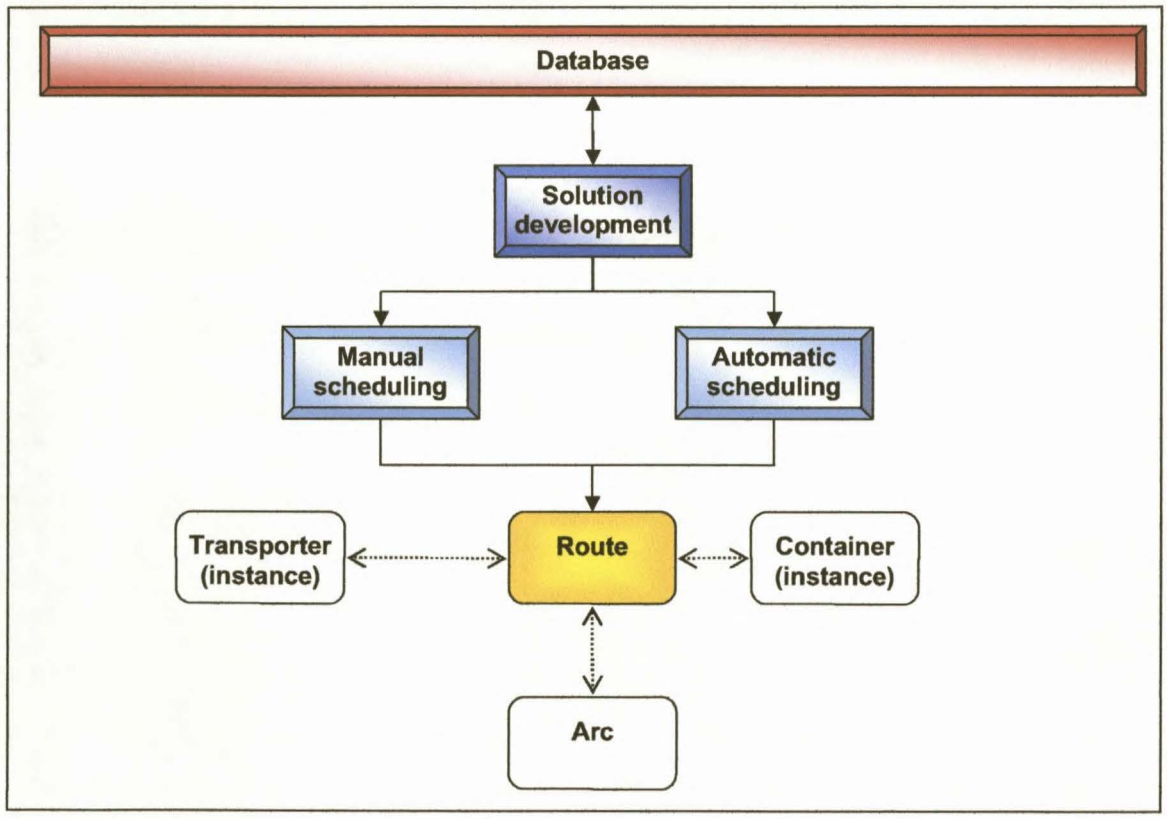



*Figure 4.3 Breakdown of solution development structure*

When a solution is created, it is stored in the database so that it can be accessed by the other program structures.

## 4.3. THE SOLUTION EVALUATION STRUCTURE

The solution evaluation structure retrieves a solution from the database and evaluates it by determining a standard set of parameters from the solution. These parameters represent the generally important aspects of a solution and can be used in specifying an objective function. The user may, however, specify the calculation of user-defined parameters if he/she values different aspects of a solution.

There are two program sub-structures used to evaluate a solution. Firstly, a reporting sub-structure generates standard reports and standard parameter values (that are useful for a variety of users), and secondly, a query sub-structure allows the user to calculate unique reports and user-defined parameter values.

The diagram below (Figure 4.4) illustrates a structure breakdown of the solution evaluation structure:

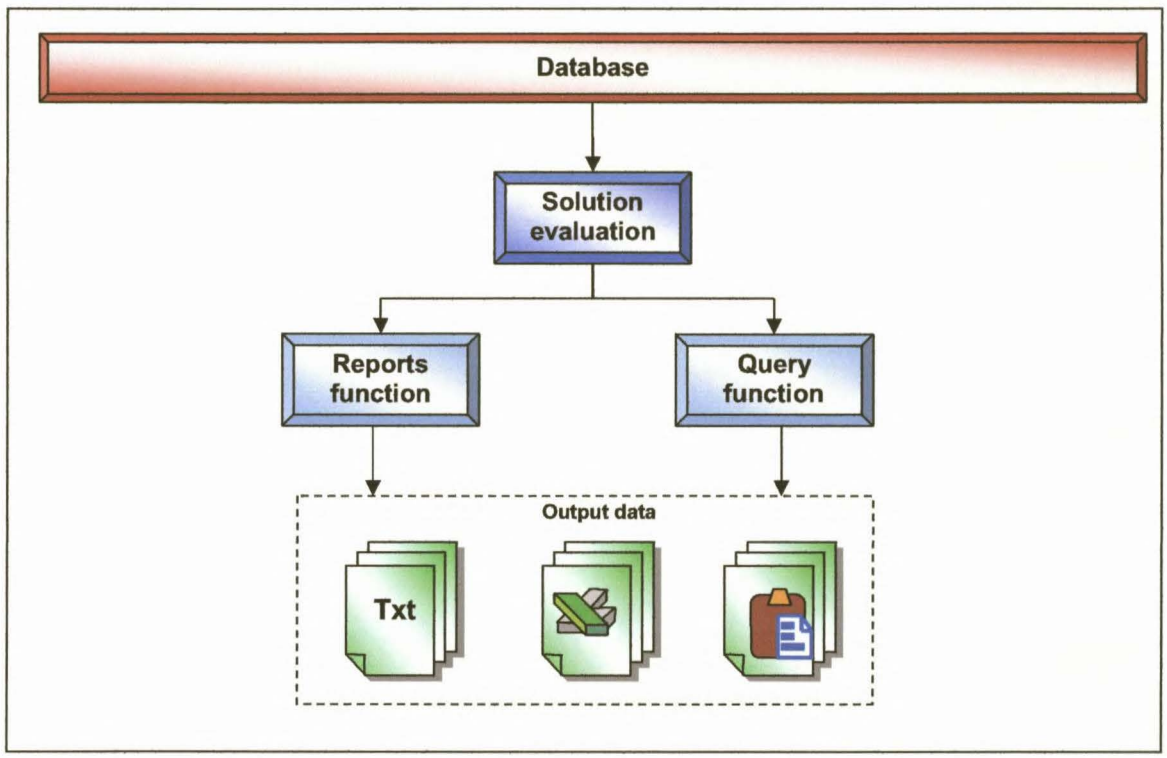



*Figure 4.4 Breakdown of Solution Evaluation Structure*

As depicted in the diagram, both sub-structures can export the output data to text files, MS Excel or the Windows Clipboard. This allows the user to analyse the output in more depth using the data analysis structure and MS Excel. The user can also paste the report in the MS Windows application of his/her choice (e.g. MS Word) by loading the report into the MS Windows clipboard.

## 4.4. THE SCENARIO SIMULATION STRUCTURE

The scenario simulation structure enables the user to evaluate different scenarios and scheduling algorithms. The effect of adding or removing transporters, arcs, vertices or containers (or even loading rules) can be simulated. The different scheduling algorithms can then be tested against these scenarios to determine which algorithm develops the best solutions under a chosen objective function.

The diagram below (Figure 4.5) illustrates how the scenario simulation structure interacts with the external and internal data structures:

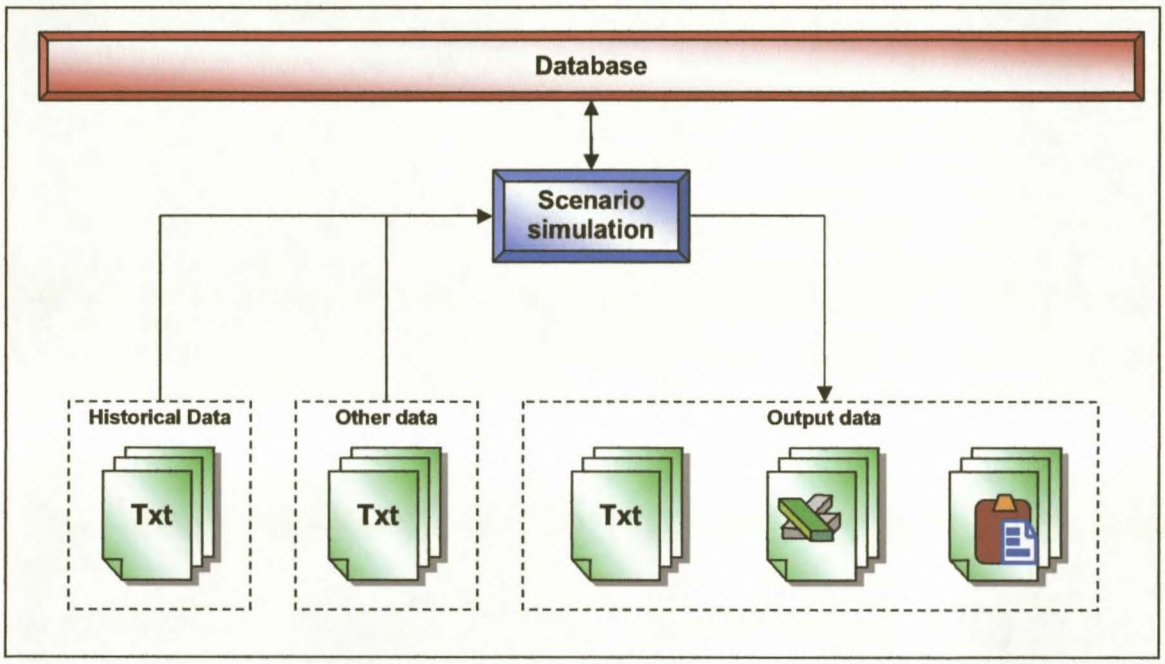



*Figure 4.5 Breakdown of scenario simulation structure*

The simulation model setup is stored in the database together with the problem definition. The scenario simulation structure allows for external data (stored in text files) to be used as part of a simulation model. (Therefore historical data and specific distributions that are not available in the software tool can be included in the simulation model.)

After a simulation is completed, parameters are calculated. These can be exported to text files, MS Excel or the MS Windows clipboard. As with the solution evaluation structure, the external output data can be analysed further with MS Excel and the data analysis structure.

When the last simulation run is executed, the solution is temporarily stored in the database and can be analysed by the simulation evaluation structure.

## 4.5. THE DATA ANALYSIS STRUCTURE

The purpose of the data analysis structure is to analyse data stored in text files statistically. This enables the user to develop simulation models that are better representations of the real-world problems so that more accurate estimations can be made.

It also allows for the statistical analysis of the output data of the scenario simulation and solution evaluation structures. The user can, for example, examine what distribution is formed by the late arrival of containers.

The data analysis structure is not restricted to the generic problem model and can be used for other purposes. Figure 4.1 on page 57 can be redrawn to depict the data analysis structure as a component separated from the other structures.

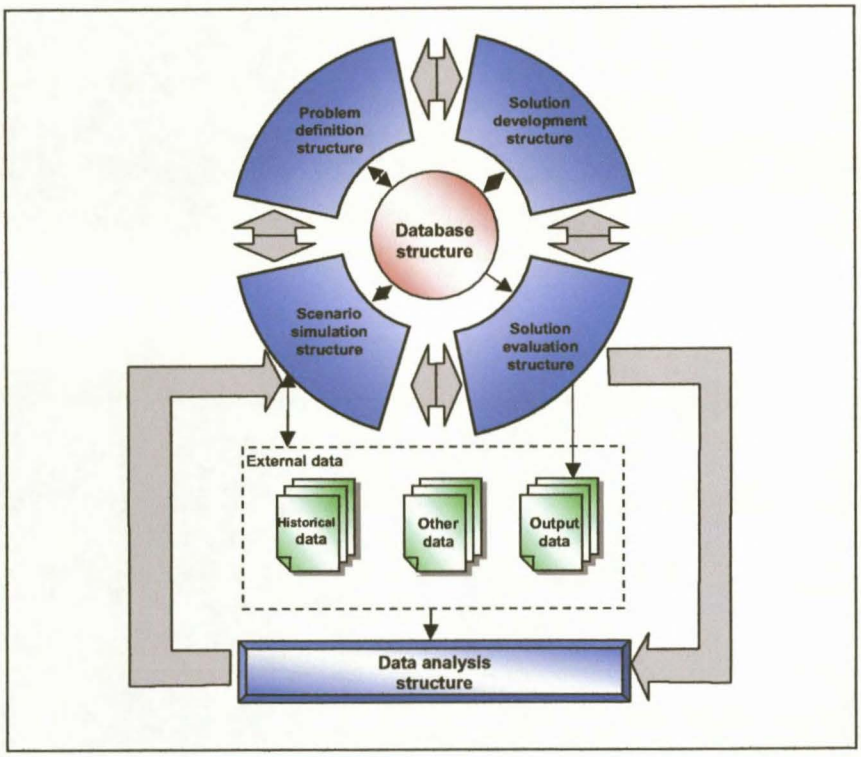



*Figure 4.6 Software tool redrawn with data analysis structure as an external structure*

The figure above shows the actual data flow between the various structures as the black arrows, while the grey arrows represent the logical process flow.

## 4.6. THE DATABASE STRUCTURE

The purpose of the database is to store an instance of the generic problem together with the solution obtained for the problem. The various entities of the problem and their associations with each other are stored in separate tables in the database.

Since the problem instance is stored in the database, the database integrates the various structures of the software tool and enables them to communicate with each other about the same problem instance. This integration function can be clearly seen on the previous page in Figure 4.6.

The database also stores information about the simulation model (if the user has developed such a model).

A more detailed discussion of the database design is given in Section 5.6. on page 89 and Appendix C.

## 4.7. CHAPTER SUMMARY

The conceptual design of the software tool was described in this chapter. The software tool consists of four integrated program structures that communicate through a data structure. These structures are sufficient to define, solve, analyse, simulate and store an instance of the generic problem. A fifth program structure allows for the statistical analysis of data so that accurate simulation scenarios can be developed and so that the output of the simulations can be analysed. The design of the software tool is described in more detail in the following chapter.

# Chapter 5 Detailed Description of Software Tool

This chapter contains a detailed description of the software tool of which the conceptual design was discussed in the previous chapter. The software tool will be discussed according to the main structures of the tool: the definition structure, the solution development structure, the solution evaluation structure, the scenario simulation structure, the data analysis structure and, lastly, the database structure. The heuristics developed to solve the transporter scheduling problem are described on page 69 in Sub-section 5.2.3.

## 5.1. THE PROBLEM DEFINITION STRUCTURE

In the previous chapter it was stated that the following five program sub-structures were used to define an instance of the generic problem in the software tool: transporter template definition, transporter definition, container template definition, container definition and transportation network definition. This section gives a detailed discussion of the development of the five sub-structures.

### 5.1.1. Transporter template definition sub-structure

This structure allows for the definition of transporter categories called templates. A transporter template contains information about a transporter type. This information includes the running cost (per kilometre) of the transporter type as well as its average transit speed.

Once a transporter template has been defined and stored in the database, it cannot be deleted unless there are no instances of the template. This ensures that a transporter cannot lose information derived from its template.

Since the cargo capacity restrictions of the transporters are defined by the transporter templates, the transporter template definition structure accommodates the definition of these rules. The loading rules are checked before a transporter template can be added or updated to ensure that the loading rules do not contradict each other.

### 5.1.2. Transporter definition sub-structure

After a transporter template has been defined, instances of the template can be created. The unique information of each instance is defined in the transporter definition structure, which includes the following information:

- The name of the transporter (e.g. a number plate)
- the initial mileage the transporter had before entering the system
- the time when the mileage was taken
- the time the transporter entered the system
- the vertex where the transporter entered the system
- the time the transporter leaves the system (i.e. the halt time of the transporter).

Once a transporter has entered the system and has been assigned to one or more movements, the transporter cannot be deleted. The transporter can, however, be removed from the system by setting the halt date to an appropriate value so that the transporter cannot be assigned to any other movements.

### 5.1.3. Container template definition sub-structure

The container template definition structure allows for the definition of container templates. The template stores information regarding the container type. Accordingly, once a container instance has been created, the corresponding template cannot be deleted unless all instances of the template are deleted as well.

If a new container template is created, the loading rules of all the transporter templates must be adapted so that container instances of the new template can be loaded onto the transporters.

### 5.1.4. Container definition sub-structure

Container instances are defined in the container definition structure. In order to define a container the following attributes of a container instance are set:

- the container's name
- the release vertex
- the destination vertex
- the release time
- the destination time
- the container's penalty rate
- the income received for transporting the container.

The container definition structure sets the default destination time and penalty rate of an instance as derived from the release vertex, but allows the user to change the values of these attributes. The structure also ensures that a container cannot be released from and destined for the same vertex. After a container has been assigned to one or more movements, the software tool will protect the user from deleting the container before unassociating it from all movements. This ensures that the solution integrity is not lost when a container is deleted.

### 5.1.5. Transportation network definition sub-structure

The transportation network definition structure enables a user to create a transportation network consisting of vertices and arcs. In this structure the user enters appropriate values for the following vertex attributes:

- the name of each vertex
- the default penalty rate for containers released at the vertex
- the default slack time for containers released at the vertex.

The user defines the connectivity between the vertices by creating a set of arcs. This is done by simply specifying the direct distance between a newly-created vertex and every other vertex in a forward and backward direction. If the forward and backward distances differ, an unsymmetrical graph will be created.

After the set of arcs has been created, the structure will evaluate the proposed graph to ensure that it is a connected graph. If one or more vertices are not reachable, the user will be prompted to correct the graph.

The structure calculates and stores all pairs of shortest paths and distances (among the different vertices) in the database when entering a valid graph. The shortest paths and distances are repeatedly used by the scheduling algorithms. Floyd's algorithm is used to calculate the shortest paths and distances in the graph. (Nemhauser et al. [62] give a discussion on Floyd's algorithm.)

## 5.2. THE SOLUTION DEVELOPMENT STRUCTURE

As stated in the previous chapter, the purpose of the solution development structure is to create valid solutions for the problem instance defined by the definition structure. Two program sub-structures were used for solution development: one that allows the user to create a solution manually, and a second that automatically creates a solution according to a chosen solution algorithm. This section gives a detailed discussion of the two program sub-structures.

### 5.2.1. Manual scheduling sub-structure

As stated above, the manual scheduling sub-structure enables the user to develop a solution to the problem. In order to solve the problem, the user has to create new movements and associate the undelivered containers with these movements so that they will arrive at their destinations. It is useful to examine the problem from two angles while developing a solution.

Firstly, the user can view the problem from the container viewpoint by examining the list of undelivered containers. A list of movements is created for each container, showing all the movements with which the container may be associated. The user can use these movements to move the undelivered container to its destination, or create new movements with which to move the container. The user can also use a combination of new and old movements to move the container.

Secondly, the user can view the problem from the transporter viewpoint by examining the movements with which a transporter is associated. A list of containers is then created for each movement, showing which containers are currently associated with the movement and which other containers can still be associated with the movement. The user can then associate these containers with the current movements or create new movements for a transporter with which containers can be associated.

The manual scheduling sub-structure enables the user to examine both viewpoints simultaneously while developing a solution. The structure also protects the user from creating an invalid solution. The following mechanisms were implemented to ensure that the user creates a valid solution:

- A new movement cannot be created for a transporter in such a way that the transporter leaves a vertex before it has arrived at the vertex and the delay time has elapsed.
- If a transporter has to be assigned to a movement that departs from a vertex which is not the last arrival vertex of the transporter, movements will automatically be created

to move the transporter from its last arrival vertex to the new departure vertex. Dijkstra's algorithm was used to find the shortest path on which the transporter should be moved from its last arrival vertex to the new departure vertex.

- A container will not be allowed to be assigned to a movement if the movement departs from a vertex which is not the container's last vertex. Similarly, a container will not be allowed to be assigned to a movement that departs from a vertex before the container arrives at that vertex.
- If the user attempts to assign a container to a movement, the loading rules of the transporter will be examined to test if the assignment of the container violates the loading capacity of the transporter. If a loading rule is violated, the user will not be allowed to assign the container to the movement.
- When a container reaches its destination vertex, the container cannot be assigned to any further movements.
- When a container is unassigned from a movement, the container will also be unassigned from all the future movements to which the container was assigned. This ensures that a container cannot be assigned to a movement from A to B and to another movement from C to D, without being assigned to an intermediate movement from B to C.
- Similarly, when a movement is deleted, the future movements of the transporter are also deleted so that the transporter cannot be assigned to a movement from A to B and to another movement from C to D, without being assigned to an intermediate movement from B to C.
- When a movement is deleted, the containers associated with the movement are unassociated from that movement and from all future movements.

These mechanisms ensure that any solution created by the user is a valid solution, although there is no guarantee of an optimal solution or even a good solution.

### 5.2.2. Automatic scheduling sub-structure

The automatic scheduling sub-structure is used to automatically create movements and assign containers to the movements so that a solution to the problem is generated. This structure is required for larger problems since it becomes time consuming for the user to schedule a large number of transporters and it is difficult for the user to create a solution that optimises a given objective function.

The automatic scheduling sub-structure contains two heuristic algorithms that can create solutions to the problem. These algorithms use the same mechanisms that protect a user from creating an invalid solution.

The user can use the scenario simulation structure to evaluate the solution algorithms according to a chosen objective function and problem configuration. The solution algorithms are discussed in the following sub-section.

### 5.2.3. Solutions algorithms developed

Two heuristics were developed to generate solutions to the problem. Both have the same form and can be simplified to the following steps:

1. Delete the future part of the current solution (if a current solution exists).
2. Create a list of containers that have not been delivered to their destination and prioritise them according to some rule.
3. **Repeat** for every container in the list:

    **Repeat**:

    Use the current transporters available in the problem and attempt to route the container to its destination using the current movements in the solution and evaluate this routing. Attempt to route the container to its destination by creating new movements and evaluate this routing. Compare the two alternatives, and using the best option, move the container to the next vertex.

    **Until** the container has reached its destination

    **Until** the list is empty (i.e. all container have been scheduled)
4. End

Since the algorithm is run at a certain point in time, it may well be that a part of the current solution to the problem (if there is a current solution) may not be deleted without losing the actual state of the real-world system. If a transporter has already departed on a movement by the time the algorithm is executed, that movement may not be deleted, since the transporter is not at the origin vertex any longer. Similarly, the containers assigned to that movement may not be unassigned from the movement, since they have already been loaded onto the transporter. The first step of the algorithm is therefore to delete the appropriate part of the current solution.

As can be seen from the outline above, the heuristics have two parts that significantly influence the performance of the heuristic: firstly, deciding how to prioritise the containers, and secondly, deciding whether it is better to use the current movements or to create new movements to move a container.

The first decision of the solution algorithms is rather simple: if the containers' late time is important they can be ranked according to the earliest destination time; otherwise, if the penalty cost is of concern they can be ranked according to the highest penalty that will be incurred by a certain date.

The second decision is more difficult to define, since it involves evaluating whether to send the container along existing movements, to create new movements, or to combine existing and new movements.

The second decision therefore also involves creating an evaluation rule as well as a method for creating sensible new movements. The evaluation rule can be as simple as using the earliest arrival time of the container or more complex, so that the running cost of the transporters are also incorporated.

It is, however, more difficult to find a method for creating sensible new movements. It is difficult to create sensible new movements because there is a whole fleet of transporters that can be used to move the container, and each of the transporters can move along various paths through the transportation network while taking the container from its release vertex to its destination vertex. Furthermore, more than one transporter can be used to move the container so that one transporter moves the container a certain distance, while others carry it for the remaining distance. As stated earlier, the container can also be moved partially using existing movements and partially by creating new movements.

The movement creation was implemented by routing the container with each transporter from its last arrival vertex (i.e. the vertex to which the container was last routed in the existing solution), to its destination vertex along the shortest path in the transportation network. Routing the container along the shortest path in the transportation network does not necessarily imply that a better solution will be guaranteed, but it seems sensible to route the container along the shortest path rather than along a random path in the transportation network. Moving the container along a shorter path is more sensible because the running cost of the transporter is lower, and the time required for moving the container is probably

shorter. (Again a shorter time is not guaranteed by the shortest path, since the shortest path may involve more stop-overs and thus more delay times.)

The routes with the different transporters are then evaluated and the best transporter and its movements are chosen and compared to routing the container with existing movements. Once the best option has been chosen, the container is moved along a single movement (i.e. transported one vertex on) by means of the chosen option. The last arrival vertex of the container is thus changed. After a container has been transported along a single movement the whole process is repeated until the container has reached its destination. This allows the algorithm to consider whether new or old movements should be used from the newly-changed last arrival vertex to route the container further.

After the container has been routed to its destination the next container in the list is selected and routed, until every container has been routed to its destination.

One aspect that has not been included in the discussion above is the possibility that a container cannot be routed to its destination by means of the existing movements. If this is the case, then new movements must be created. It may be possible that a container cannot be assigned to a movement or a transporter (since a loading capacity restriction is violated, or the transporter is halted), in which case the movement or transporter is temporarily removed from the graph while the different options for the container are re-examined.

Since the two most important aspects of the Autocarriers problem are the service level and the total cost, two heuristics were developed to find optimum values for these parameters. The heuristics were developed within the framework of conditions discussed on page 46 to 47. A detailed discussion of each of the heuristics developed follows below.

An important assumption made in the development of the algorithms is that the cost of reloading containers between transporters is negligible. The cost of reloading containers may be significant and can be incorporated in the scheduling model. However, it was left to further research to develop algorithms that minimises the reloading of containers.

(a) Algorithm 1: Scheduling for a maximum service level (MSL)

The purpose of the maximum service level (MSL) heuristic is to deliver the containers as soon as possible, and thereby maximise the service level[1] and minimise the overall late time of containers. As a consequence, container penalty cost is decreased, although transporter running cost may be raised.

The rule for evaluating the different routing options was to use the routing that delivered the container as early as possible.

An outline of this heuristic can be seen on the next page.

[1] The service level is the percentage of containers that are delivered on time. Refer to section 5.3.1. (a) on page 76 for the formal definition.

SCHEDULING FOR A MAXIMUM SERVICE LEVEL:

1 Delete all movements that have a start time > $T$ (where $T$ is the time at which the algorithm is executed).

2 Create a list, say $L_C$, of undelivered containers, sorted according to earliest destination time.

3 **For** each container in $L_C$:

3.1 Let $L_T$ be the list of all available transporters in the current problem. (This excludes transporters that have been halted.)

3.2 **Start Repeat**: Consider the highest ranked container in $L_C$.

3.3 Find the container's last arrival vertex and last arrival time, say $t_C$.

3.4 Let $G$ be the time-dependent graph of existing movements departing later than max($T$; $t_C$).

3.5 Use Dijkstra's algorithm to find a path of earliest delivery time in $G$ from the container's last arrival vertex to its destination vertex. (This is the earliest delivery time of the container using existing movements.) If the container can be routed to its destination without being late, then go to step 3.9. If a container will be late, or if no path can be found, then go to the next step.

3.6 **For** each transporter in $L_T$:

3.6.1 Find the transporter's last arrival vertex.

3.6.2 Use Dijkstra's algorithm to find a shortest path in the transportation network from the transporter's last arrival vertex to the container's last arrival vertex (to move the transporter to the container's current position). Also find a shortest path in the transportation network from the container's last arrival vertex to its destination vertex (to move the transporter to the container's destination). (Note: **If** the transporter arrives at the container's last arrival vertex before the container, **then** the transporter must wait at the vertex until the container arrives at the vertex.)

3.6.3 Find the arrival time if the transporter is to be routed along the two paths found in the previous step.

3.6.4 **End For** (Keep on evaluating each transporter.)

3.6.5 Examine the arrival times of the transporters and find the transporter that has the earliest arrival time at the container's destination. (This is the earliest delivery time of the container using new movements.)

3.7 Compare the earliest delivery time using new movements (found in step 3.6.5) with the delivery time using existing movements (found in step 3.5). **If** the delivery time found in step 3.6 is earlier than that of step 3.5. **then** go to the next step, **else** go to step 3.9.

3.8 Create new movements to move the appropriate transporter to the container's last arrival vertex (This was the first path in step 3.6.2.) Create a single movement that moves the container along the second path found in step 3.6.2 (to move the container towards its destination). Attempt to assign the container to the last movement created. **If** a loading capacity is violated **then** remove all the movements created in this step, remove the transporter from $L_T$ and go to step 3.6; **else** go to step 3.10.

3.9 Attempt to assign the container along a single movement in the path found in step 3.5. **If** a loading capacity is violated then remove the movement from $G$ and go to step 3.5; **else** go to step 3.10.

3.10 **If** the container has been routed to its destination **then** remove the container from list $L_C$.

3.11 **Repeat End** (Keep on scheduling container until it is routed to destination.)

**End For** (Keep on scheduling the containers in list $L_C$ until the list is empty.)

4 End.

(b) Algorithm 2: Scheduling for a minimum total cost (MTC)

The purpose of this heuristic is to deliver the containers at a minimum total cost which is the sum of the total penalty cost and the total running cost. The total penalty cost is the cost incurred by delivering containers late, while the total running cost is the cost incurred by transporters moving. These costs are defined in the next section on page 76.

Rather than delivering containers as soon as possible, the delivery of containers can be delayed so that the transporters are used more effectively. The minimum total cost (MTC) heuristic weighs the penalty cost incurred by the late delivery of a container against the running cost of the transporter delivering the container.

There are two main differences between the MTC heuristic and the MSL heuristic: the way in which they prioritise the undelivered containers and the method used for evaluating paths. The priority rule for ranking the containers is the potential penalty cost incurred by the containers at the time the heuristic is executed. The potential penalty cost of the $i^{th}$ container, $p_i$, is calculated using its penalty rate of the container, $r_i$, and the time window starting at the time the algorithm is executed, $T$, to the destination time of the container, $D_i$, so that

$$p_i = r_i \times (T - D_i). \qquad (5.2.3.\ A)$$

The method for evaluating different paths is implemented as follows: each path results in two costs: a cost incurred by the containers being late, and a cost incurred by moving a transporter from its current location to the container's destination via the container's current position. A control parameter $\delta$ ($\delta > 0$) is used to scale the penalty cost incurred by a single container to reflect the total penalty cost incurred by all the containers associated with the path. When a new path is examined, this scaled single container penalty cost is added to the transporter running cost. The sum of these costs is then a reflection of how expensive a new path is. The cost of a new path can be evaluated against the cost of sending the container along an existing path. To make a fair comparison, the penalty cost incurred by sending the container along an existing path is also scaled.

The effectiveness of the MTC heuristic is dependent of the $\delta$ value chosen. The optimal $\delta$ value may differ from problem to problem, and it is dependent on the objective function chosen. It is recommended that $\delta$ is at least in the same order of magnitude as the number of containers that can be assigned to a transporter. If the container penalty rates are very low then $\delta$ can be increased in order to prompt the heuristic to deliver the containers earlier.

An outline of the MTC heuristic is given on the next page, whereafter the solution evaluation structure is discussed. Solution evaluation structure is used to evaluate a solution created by the solution development structure.

## SCHEDULING FOR A MINIMUM TOTAL COST:

1 Delete all movements that have a start time > $T$ (where $T$ is the time at which the algorithm is executed).

2 Create a list, say $L_C$, of undelivered containers, sorted according to the potential penalty (i.e. $p_i$) incurred by the container at time $T$.

3 **For** each container in $L_C$:

3.1 Let $L_T$ be the list of all available transporters in the current problem. (This excludes transporters that have been halted.)

3.2 **Start Repeat**: Consider the highest ranked container in $L_C$.

3.3 Find the container's last arrival vertex and last arrival time, say $t_C$.

3.4 Let $G$ be the time-dependent graph of existing movements departing later than max($T$; $t_C$).

3.5 Use Dijkstra's algorithm to find a path of earliest delivery time in G from the container's last arrival vertex to its destination vertex. If the container can be routed to its destination without being late, then go to step 3.9. If a container will be late, determine the container penalty cost, $C_{p[existing]}$, and go to the next step. If no path can be found then go to the next step.

3.6 **For** each transporter in $L_T$:

3.6.1 Find the transporter's last arrival vertex.

3.6.2 Use Dijkstra's algorithm to find a shortest path in the transportation network from the transporter's last arrival vertex to the container's last arrival vertex (to move the transporter to the container's current position). Also find a shortest path in the transportation network from the container's last arrival vertex to its destination vertex (to move the transporter to the container's destination). (Note: **If** the transporter arrives at the container's last arrival vertex before the container, **then** the transporter must wait at the vertex until the container arrives at the vertex.)

3.6.3 Find the arrival time for the transporter if it is routed along the two paths found in the previous step. Let $C_n$ be the weighted delivery cost associated with routing the container with transporter $n$: $C_n = C_r + \delta C_p$, where $C_r$ is the running cost incurred by the transporter moving along the two paths in step 3.6.2; $C_p$ is the penalty cost incurred by delivering the container at the arrival time found in step 3.6.2. and $\delta$ is the weighting factor.

3.6.4 **End For** (Keep on evaluating each transporter.)

3.6.5 Examine the costs of transporting the container with the various transporters and find the transporter that has the lowest weighted cost.

3.7 Compare the delivery cost using new movements, i.e. the $C_n$ (found in step 3.6.5), with the delivery cost using existing movements, i.e. the $C_{p[existing]}$ (found in step 3.5). **If** $C_n < \delta C_{p[existing]}$ **then** go to the next step, **else** go to step 3.9.

3.8 Create new movements to move the appropriate transporter to the container's last arrival vertex. (This was the first path in step 3.6.2.) Create a single movement that moves the container along the second path found in step 3.6.2 (to move the container towards its destination). Attempt to assign the container to the last movement created. **If** a loading capacity is violated **then** remove all the movements created in this step, remove the transporter from $L_T$ and go to step 3.6; **else** go to step 3.10.

3.9 Attempt to assign the container along a single movement in the path found in step 3.5. **If** a loading capacity is violated **then** remove the movement from $G$ and go to step 3.5; **else** go to step 3.10.

3.10 **If** the container has been routed to its destination **then** remove the container from list $L_C$.

3.11 **Repeat End** (Keep on scheduling the container until it is routed to its destination.)

**End For** (Keep on scheduling the containers in list $L_C$ until the list is empty.)

4 End.

## 5.3. THE SOLUTION EVALUATION STRUCTURE

The solution evaluation structure evaluates a solution stored in the database. A solution can be evaluated according to specific parameters (such as costs incurred), or in practical terms (such as the schedule for the transporters). Some of the specific parameters that can be calculated reflect the important aspects of a solution mentioned in Section 3.2.1. on page 45.

The solution evaluation structure is implemented using two program sub-structures. The first sub-structure, the reporting structure, allows for the standard evaluation of a solution, while in the second sub-structure, the query structure, the user can request a unique evaluation.

### 5.3.1. Reports Sub-Structure

The reports sub-structure can generate the following five different types of reports: A period performance report; a daily performance report, a vertex report; a transporter report and a container report. Each of these reports is discussed below:

(a) Period performance report

The period performance report is useful for evaluating a solution over a given time period. Different companies have different business strategies and will therefore have different objective functions for evaluating a solution. It was decided to calculate the six parameters so that the important aspects of a solution are determined as mentioned on page 45 in Section 3.2.1. These parameters are:

- the total container penalty cost
- the total transporter running cost
- the total cost incurred
- the service level of the solution
- the average late time of the containers
- the average transporter efficiency.

The total container penalty cost is the sum of all penalties incurred by containers that are delivered later than their destination time so that the

$$\text{Total Penalty Cost} = \sum_{i=1}^{n} r_i \times \max(0; A_i - D_i), \qquad (5.3.1.\text{ A})$$

where $r_i$ is the penalty rate of container $i$, $A_i$ the (actual) arrival time of container $i$ at its destination, $D_i$ is the (required) destination time of container $i$ and $n$ is the number of containers released in the period evaluated.

The penalty rate and destination time of each container is specified when the problem instance is defined. When the container is delivered late, a penalty is paid at the appropriate rate on the difference between its arrival time (at the destination vertex), $A$, and its destination time, $D$ (which is the latest acceptable delivery time).

The total transporter running cost is the cost incurred by transporters moving between vertices as described by

$$\text{Total Running Cost} = \sum_{i=1}^{n} c_i \times d_i \,, \qquad \text{(5.3.1. B)}$$

where $c_i$ is the running cost (in R/km) of the transporter used for the $i^{th}$ movement in the period and $d_i$ is the distance (in km) of that movement. $n$ represents the number of movements departing in the period evaluated.

The total cost incurred is the sum of the total penalty cost and total running cost, so that the

$$\text{Total Cost} = \text{Total Penalty Cost} + \text{Total Running Cost}\,. \qquad \text{(5.3.1. C)}$$

The service level of a solution is defined as the percentage of containers that are delivered on time, so that the

$$\text{Service Level} = \frac{\text{Number of Containers Delivered on Time}}{\text{Total Number of Containers}} \qquad \text{(5.3.1. D)}$$

The average late time is defined as the average number of hours that a container will be delivered late, given that it will not be delivered on time. This average late time therefore does not incorporate containers that are delivered on time. The average late time is described as

$$\text{Average Late Time} = \frac{\sum_{i=1}^{n} (A_i - D_i)}{n}, \qquad \text{(5.3.1. E)}$$

where $A_i$ and $D_i$ are the respective arrival and destination times for the $i^{th}$ late container and $n$ is the number of late containers in the period evaluated. The arrival and destination times are explained below (5.3.1.A).

The average transporter efficiency is an indication of how efficiently movements have been created in a solution. It is therefore also a reflection of how efficiently the transporters have been used, although it should not be confused with the traditional definition of equipment utilisation. Average transporter efficiency is defined as the ratio of the actual number of

containers transported to the number of containers that could be transported on a movement, weighed according to the running cost incurred. The transporter efficiency is described as

$$\text{Average Transporter Efficiency} = \sum_{i=1}^{n} \frac{L_i \times c_i}{L_{i\max}} \Big/ \sum_{i=1}^{n} c_i \,, \qquad \text{(5.3.1. F)}$$

where $L_i$ is the number of containers assigned to movement $i$, $c_i$ is the running cost[2] (in rand) incurred by the transporter traversing movement $i$, and $n$ is the number of movements starting in the period evaluated. Here $L_{i\max}$ is the total number of containers that can be assigned to the transporter traversing movement $i$ (by using the last loading rule applied while assigning a container to the movement). The maximum number of containers that can be assigned to a transporter may depend on the loading rule used.

These six parameters are useful for understanding what the broad trends are in a solution, in other words, how efficiently the transporters are utilised, what the service level is, and what costs are involved in obtaining the service level. One is able to see how efficiently the transporters are being used and what proportion of containers is delivered late, as well as the costs associated with the solution. The trends can be examined further over a time period using the daily performance report.

(b) Daily performance report

The daily performance report calculates the same parameters as the period performance report, but over 24 hour periods so that the daily performance of a solution can be determined.

This report is useful for seeing practical trends in a company's history (e.g. that more containers are delivered late towards the end of a month). These trends may be of a positive or negative nature and may prompt management to take corrective steps (e.g. to hire an extra transporter during the last week of a month to increase the service level).

The daily performance report is also useful to determine the warm-up time for scenario simulations. The daily calculations can be plotted so that the start of the steady state of the system can be determined.

[2] The cost associated with a route in this case, is the running cost incurred by the transporter. The cost of the route is thus a cost in rand (i.e. the running cost rate of the transporter [R/km] multiplied by the distance of the route [km]).

(c) Vertex report

This report is not useful in determining the quality of a solution but has a great practical value. The report focuses on a vertex (chosen by the user) and gives a list of transporters arriving and departing from the vertex as well as a list of containers that should be released from and delivered to the vertex.

(d) Transporter report

The transporter report, like the vertex report, is of practical value to the user. The report lists the various movements and destinations of a specific transporter, as well the containers that should be loaded onto and off the transporter for each movement.

(e) Container report

The container report is useful for a more in-depth evaluation of a solution. The report lists all the details of containers released from a chosen release vertex and transported to a chosen destination vertex. The user is thus able to create histograms (with the data analysis structure) of the container late times and other details. The user is also able to select all vertices as destination or release vertices so that the late time distribution of all containers released from a vertex or destined for a vertex can be estimated.

### 5.3.2. Query sub-structure

The query sub-structure gives the user direct access to the database by means of SQL queries. This enables the user to create self-defined reports and thereby evaluate a solution according to a unique objective function.

The user can create simple SQL queries that retrieve information from the database, or aggregate SQL queries that can perform calculations on the data in the database. These functions include counting, summation and averaging. The user can also create operational SQL queries that can add, edit or remove data in the database. Operational queries allow the user to make many changes quickly (e.g. to change the names of all the containers in the database).

The output generated by the reports sub-structure and the query sub-structure can be exported from the software tool to either text files or MS Excel. Exporting the output to text files enables the user to evaluate the output by means of the data analysis structure, while

exporting the data to MS Excel gives the user access to a wide variety of functions and tools to analyse the data.

The simulated solutions developed in the scenario simulation structure can be examined with the reports sub-structure or with the query sub-structure. The scenario simulation structure is discussed in the following section.

## 5.4. THE SCENARIO SIMULATION STRUCTURE

As stated in the previous chapter, the purpose of the scenario simulation structure is to facilitate the simulation of various configurations of an instance of the generic problem. There are two motivations for simulating configurations of an instance of the generic problem: firstly, to discover which heuristic creates the best solutions for a given instance of a problem subject to a specific objective function, and secondly, to evaluate the influence of various changes to a problem.

The ability to execute simulations in the software tool is described in this section. The process by which a simulation model is created is discussed, whereafter the execution of a simulation is described.

### 5.4.1. Creation of simulation model

The creation of a simulation model involves three steps. Firstly, all the entities of the problem (except the container entities) must be defined. (This can be done by means of the problem definition structure.) If the software tool has been implemented by a client, the client's current problem definition can be used.

Secondly, the time period of the simulation, the number of runs required and the scheduling heuristic must be specified.

Thirdly, the configuration by which containers enter the system must be specified. The scenario simulation structure allows containers to enter the system as group arrivals of a certain container template, released at a vertex with a certain inter-arrival time and destined for another vertex. The container entering configuration is specified as a series of container entering patterns.

Each entering pattern has the following attributes which are discussed on the next page:

1. release vertex
2. destination vertex
3. container template
4. group size
5. inter-arrival time between groups
6. notification time of group.

The release vertex is the vertex where the group of containers will be released, while the destination vertex is the vertex to which all of the containers in a group must be delivered. The group size represents the fixed number of containers in a group and can be set to 1 if group arrivals are not applicable.

The inter-arrival time between groups can either be stochastic or deterministic. A variety of distributions can be used for stochastic inter-arrival times. The deterministic inter-arrival time can be specified as daily, weekly or monthly, or alternatively, be read from a text file.

The notification time is the period between a group entering the system and the group being released. When a container (or container group) is entered in the system it may not yet be ready to be released (and thus not be ready for pickup), but the release time of the container is available. This allows a scheduler (manual or automatic) to send a transporter to the container's release vertex in advance. The shorter a container's notification time, the more difficult it becomes to find an optimal schedule since the scheduler's knowledge of the future is limited. (The notification time is, in practical terms, the ability of a manufacturer to forewarn Autocarriers that a vehicle will be ready for pickup at a certain time. Autocarriers can then adapt their carrier schedule accordingly.)

As with the inter-arrival times, the notification time can be deterministic or stochastic. The Beta, Erlang, Exponential, Lognormal, Normal, Rectangular and Weibull distributions can be used for the stochastic inter-arrival times and notification times. The definitions of these distributions and the methods of random variate generation are stated in Appendix A.

Table 5.1 below is an example of a container entering configuration:

| Container entering configuration | | | | | | |
|---|---|---|---|---|---|---|
| | **Release vertex** | **Destination vertex** | **Container template** | **Group size** | **Interarrival time [h]** | **Notification time [h]** |
| Pattern 1 | CT | JHB | High load veh. | 1 | Expo (5) | Fixed (24) |
| Pattern 2 | CT | JHB | Low load veh. | 2 | Expo (3) | Fixed (24) |
| Pattern 3 | JHB | CT | High load veh. | 1 | Erlang (2,3) | Normal (24,4) |
| .... | ... | ... | ... | ... | ... | ... |

*Table 5.1 Example of a container entering configuration*

After a simulation model has been defined it can be executed. The following sub-section contains a discussion of how the model is executed.

### 5.4.2. Execution of simulation model

When a simulation model is executed the following steps are carried out for every run: a list of simulated containers is created according to the entering patterns defined; this list is then sorted according to the containers' notification time; the relevant containers for every period are then entered into the system and the transporters scheduled. When the end date for the simulation model is reached, the solution is evaluated whereafter it is deleted and the next run executed. This process continues until the specified number of runs has been executed.

The diagram (Figure 5.1) below illustrates the steps that are followed when executing a simulation model:

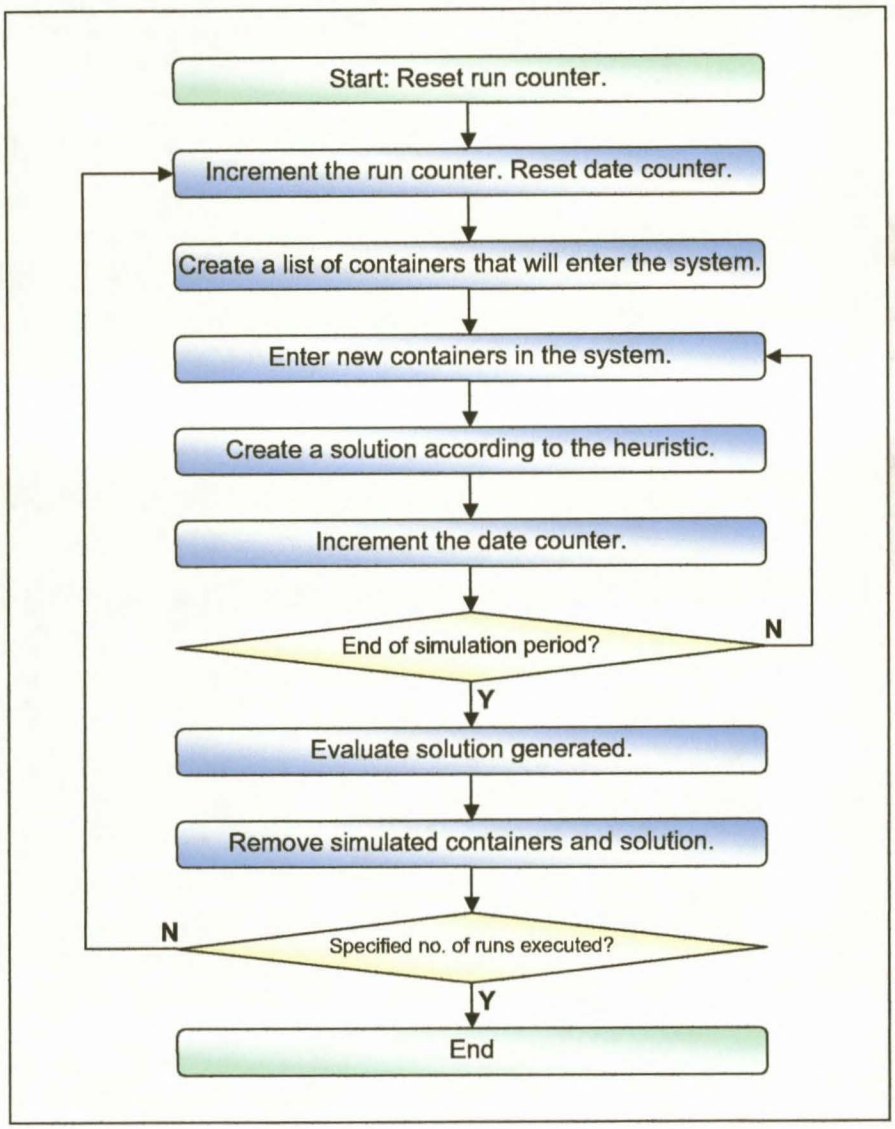



*Figure 5.1 Flow diagram of a simulation*

The two heuristics available in the automatic scheduling function can be used in the scenario simulation structure.

The period considered for evaluation can be specified by the user so that a warm-up period can be ignored if necessary. The period performance reporting function of the solution evaluation structure is used to evaluate the solution created by the simulator. The following parameters that were defined in Section 5.3.1. (a) on page 76 can be calculated:

- the total container penalty cost
- the total transporter running cost
- the total cost
- the service level
- the average late time.

Since the scenario simulation structure temporarily stores the last simulation run as a solution in the database before deleting it for the next run, the user can use the solution evaluation structure to do an in-depth evaluation of the last run in the database. The reports sub-structure or query sub-structure can be utilised for the evaluation of this single run. The solution evaluation structure, together with the data analysis structure, can be used to determine the distribution of container late time, transporter utilisation or other variables required by the user.

The data analysis structure is discussed in the next section.

## 5.5. THE DATA ANALYSIS STRUCTURE

The data analysis structure forms a simple analyser that enables the user to evaluate data stored in text files statistically. This allows for the analysis of a company's historical data and thereby the creation of accurate simulation models. Similarly, the output of a simulation can be evaluated with the data analysis structure.

The data analysis structure has the following features:

- random variate creation
- sample parameter calculation
- distribution parameter estimation
- sample histogram and distribution plots
- chi-squared goodness-of-fit testing
- arrival time conversion to inter-arrival times.

A discussion of the features of the data analysis structure follows below:

### 5.5.1. Random variate creation

The random variate creation feature can be used to create a text file containing random variates generated according to a variety of distributions. The text files can be used for a simulation model or for other purposes. The same mechanisms used in the scenario simulation structure for random variate creation were used in the data analysis structure and are listed in Appendix A.

### 5.5.2. Sample parameter estimation

The data analysis structure can be used to examine data stored in a text file. The data in the text file must be stored as numeric values separated by the ASCII line-feed and carriage-return characters (i.e. character 10 and 13 respectively), so as to form a single column of data. Non-numeric lines in the text file are ignored.

The sample mean, sample variance, sample size and maximum and minimum values can be calculated from a sample stored in a text file. The sample mean is calculated as

$$\bar{x} = n^{-1} \sum_{i=1}^{n} x_i \,, \qquad \text{(5.5.2. A)}$$

where $n$ is the sample size and $x_i$ is the $i^{th}$ observation in the file.

The sample variance is calculated as

$$s^2 = (n-1)^{-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2. \qquad (5.5.2.\ B)$$

The sample standard deviation, $s$, is the square root of the sample variance.

If the observations represent the means of samples taken from the same population, then they are asymptotically normally distributed and a confidence interval for the sample mean can be constructed. The confidence interval half-width is

$$h = t_{n-1,1-\alpha/2}\sqrt{\frac{s^2}{n}} = \frac{t_{n-1,1-\alpha/2}\cdot s}{\sqrt{n}}, \qquad (5.5.2.\ C)$$

where $t_{1-\alpha/2}$ is the upper 1 – α/2 critical point for the t-distribution with $n$ -1 degrees of freedom.

The confidence interval of the sample mean represents the interval in which the population mean can be expected with a probability of 1 – α. The range in which the population mean can be expected with the aforementioned confidence is $\bar{x} \pm h$.

### 5.5.3. Distribution parameter estimation

The data analysis structure can estimate the parameters for the following distributions: Beta, Erlang, Exponential, Gamma, Lognormal, Normal and Weibull. The methods used for the parameter estimation for each of these distributions are described in Appendix A. The parameters of all the distributions mentioned, except the Weibull and Lognormal distributions, can be estimated once the sample mean and variance have been calculated.

The Weibull distribution's parameter estimation is more complicated, since it requires the iterative estimation of the parameters $b$ and $c$ in the Weibull distribution definition

$$f(x) = \frac{cx^{c-1}}{b^c}\exp\left(-\left(\frac{x}{b}\right)^c\right), \qquad (5.5.3.\ A)$$

where $b$ is a scale parameter referred to as the characteristic life ($b$ > 0) and $c$ is a shape parameter.

The two parameters defining the variate can be estimated iteratively using the expressions

$$b \approx \left[n^{-1}\sum_{i=1}^{n} x_i^c\right]^{\frac{1}{c}} \text{ and} \qquad (5.5.3.\ B)$$

$$c \approx \frac{n}{(1/b)^c \sum_{i=1}^{n} x_i^c \log x_i - \sum_{i=1}^{n} \log x_i}. \qquad (5.5.3.\ C)$$

The following process was used to estimate the parameters $b$ and $c$: Let $b_k$ and $c_k$ be the respective $k^{th}$ estimations of $b$ and $c$ where $k \geq 1$ and $k \in \boldsymbol{Z}$. Let $c_0$ be the initial assumed value of $c$. Setting $c_0 = 1$ reduces the Weibull distribution to an Exponential distribution (which occurs frequently in a variety of practical situations). The parameters are repeatedly estimated as follows: $b_k$ is calculated using $c_{k-1}$ in expression (5.5.3. B). Once $b_k$ has been calculated, $c_k$ is calculated using $b_k$ in (5.5.3. C).

The estimation process is terminated if $c_{k+1}$ is within a certain proximity of $c_k$ or a maximum number of iterations has been reached. The proximity condition ensures that the values are continually estimated until the improvement of the estimation is insignificant. The iteration condition protects the process from an endless loop, which may result from estimating Weibull parameters from a sample that cannot be represented by a Weibull distribution. (The software tool uses a default 1% proximity range and a maximum of 20 iterations. These values were chosen to avoid excessively long calculation times on the currently available computers.)

The Lognormal distribution definition used is

$$f(x) = \frac{1}{xc\sqrt{2\pi}} \exp\left(\frac{-(\log(x/m))^2}{2c^2}\right), \qquad (5.5.3.\ D)$$

where $m$ is a scale parameter ($m > 0$) representing the median and $c$ is a shape parameter ($c > 0$) representing the standard deviation of the log of the variate.

The parameters of the lognormal distribution can be estimated using the expressions

$$m \approx \exp\left((n^{-1}) \sum_{i=1}^{n} \log(x_i)\right), \qquad (5.5.3.\ E)$$

$$c^2 \approx \left((n-1)^{-1}\right) \sum_{i=1}^{n} (\log(x_i) - b)^2, \qquad (5.5.3.\ F)$$

where $x_i$ is the $i^{th}$ observation in the text file.

### 5.5.4. Sample histogram, distribution plot and chi-squared goodness-of-fit testing

The data analysis structure has a histogram plot function, which plots a histogram of the data in the text file. The intervals of the histogram are evenly spread over the observed data range. The user can choose the number of intervals up to a maximum of 50 intervals, but the default number of intervals of the histogram used is determined by Sturges's rule

$$k = \lfloor 1 + \log_2 n \rfloor. \qquad (5.5.4.\ A)$$

where $k$ is the number of intervals and $n$ is sample size (number of observations in the text file).

After a histogram has been compiled, a variety of distributions can be fitted graphically to the histogram. This is a useful feature that can be used to estimate roughly whether the data fits a specific distribution. The chi-squared goodness-of-fit test can be used for a scientific assessment.

The chi-squared goodness-of-fit test is available in the data analysis structure for the Normal and Exponential distributions. All distributions available in the data analysis structure can be tested with the chi-squared test, but it was regarded as sufficient for the scope of this project to implement the test only for the Normal and Exponential distributions. The detailed procedure followed by the data analysis structure is described in Appendix B.

### 5.5.5. Arrival time conversion to inter-arrival times

Included in the data analysis structure is a function that can convert a series of arrival times to inter-arrival times. Historical data exported from a company's database can thus easily be converted to a format that is useful for statistical analysis and the development of a scenario model. The arrival times must be stored as a single column of data in a text file in any standard date format. Since some formats may be ambiguous it is recommended that the following format be used in the input file: *dd MMM yyyy hh:mm:ss,* where *dd* is the two-digit day of the month, *MMM* the three-letter acronym of the month name, yyyy the four-digit year, *hh* the two-digit hour, *mm* the two-digit minute and *ss* the two-digit second. The output text file contains the inter-arrival times in hours.

## 5.6. THE DATABASE STRUCTURE

As mentioned in Section 4.6. on page 63 the database structure integrates the various program structures of the software tool by enabling them to communicate about the same instance of the generic problem.

The database stores the entities of the problem instance in various tables. The following diagram (Figure 5.2) depicts an entity relationship diagram (ERD) for the most important tables in the database.

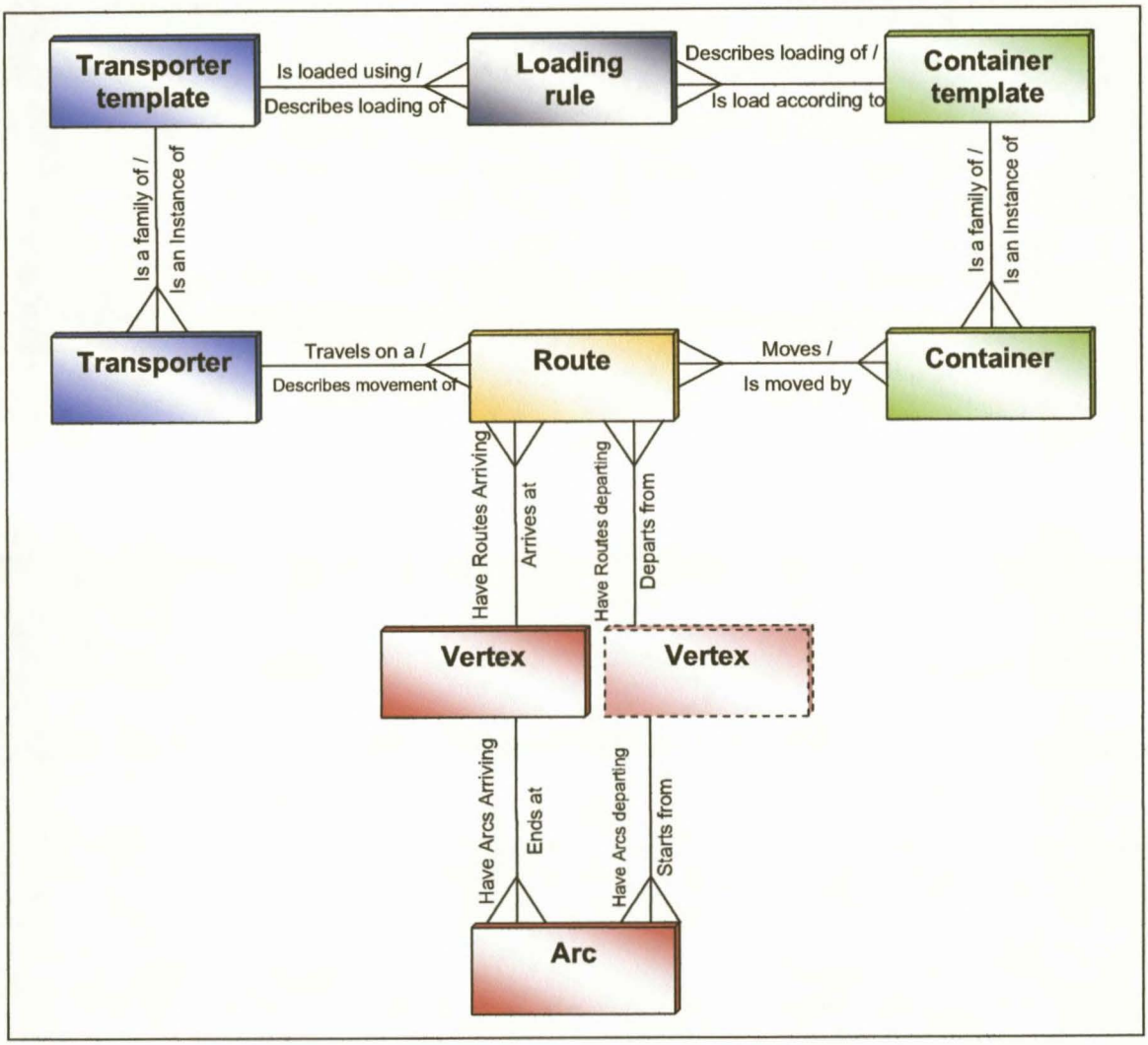



*Figure 5.2 Summary entity relationship diagram*

A complete extended entity relationship diagram (EERD) for the database, as well as a short description of each table is included in Appendix C.

## 5.7. CHAPTER SUMMARY

The design of the six structures of the software tool is described in this chapter. The definition structure uses five sub-structures to aid the user in defining an instance of the generic problem, which is then stored in the database.

The solution development structure can be used to schedule the transporters manually or automatically. The manual scheduling function protects the user from creating an invalid solution. Heuristics are used in the automatic scheduling function to create a solution for the problem defined in the definition structure. These heuristics use the same protection mechanisms to ensure that a valid solution is created.

When a solution has been developed it can be evaluated by the solution evaluation structure. There are five standard reports that are useful for a solution evaluation, but the software tool can create a custom report which is defined by a SQL query. Two of the standard reports calculate parameters that are useful for evaluating a variety of objective functions.

The heuristics in the solution development structure can be used to examine different simulated scenarios. The scenario simulation structure is useful for determining which heuristic best optimises a scenario for a given objective function. Each solution of a simulation run is evaluated using the period performance report in the solution evaluation structure.

The data analysis structure can examine input and output data stored in text files. The mean, variance and standard deviation of the data are calculated. These parameters are then used to estimate parameters for a variety of theoretical distributions. Once a theoretical distribution has been chosen a chi-squared goodness-of-fit test can be performed to evaluate whether or not the distribution describes the data in the text file adequately. Other features are also included in the data analysis structure that may be useful in a practical implementation of the software tool.

The evaluation of a variety of scenarios and the heuristics by means of the software tool is described in the following chapter.

# Chapter 6 Scenarios Evaluated with the Software Tool

The aim of this chapter is to demonstrate the value of the software tool by applying the scheduling heuristics to the Autocarriers problem. A variety of changes were made to the original Autocarriers problem and these scenarios were simulated to examine the influence of the changes. The simulation procedure that was used is discussed, whereafter the results of the different scenarios are described. The output of the various simulations is listed in Appendix D.

## 6.1. PROCEDURE FOLLOWED

The scenarios evaluated could be modelled as either a terminating or non-terminating system, depending on the information required. If the daily values of parameters are required, the model should be set up as a non-terminating system, since any day's parameter values are dependent on the previous day's parameters. Since non-terminating systems require longer runs and the simulator is relatively slow, it was decided to model the system as a terminating system.

Therefore, instead of determining daily parameters, the parameters were determined at the end of a period. That is to say, instead of determining the daily costs of a model, it was decided to calculate the total cost incurred by the algorithm as if it were used over a period of a week to schedule the transporters.

Figure 6.1 below illustrates how the system can be simulated as a non-terminating system using the batch-means approach. After the system has reached a steady state, the simulated time is broken up into batches and the parameter values determined for each batch. Although this procedure was not used in the simulations described in this chapter, the software tool allows for the batch-means approach to be used.

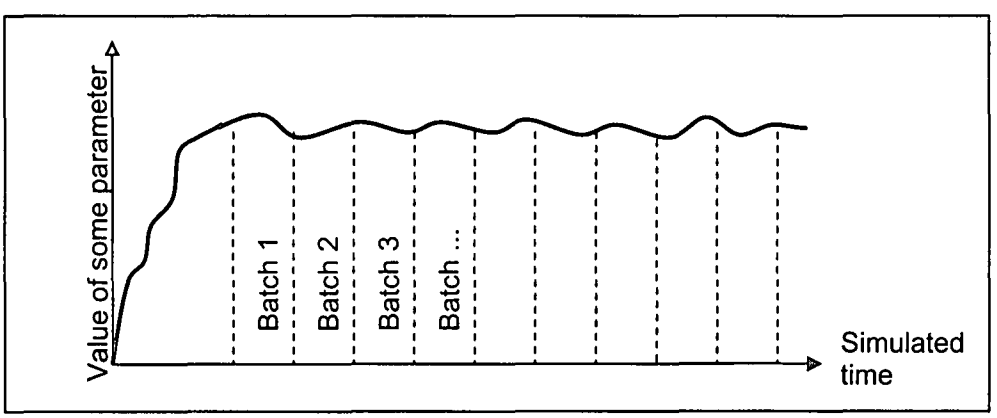



*Figure 6.1 Simulating the system as non-terminating*

The system can also be simulated as a non-terminating system and the steady state values determined using the replication-deletion approach. This approach was used in this project and is depicted in the diagram below (Figure 6.2). Using this method involves making various replications and determining the steady state parameter values for one batch in each replication.

The replication-deletion approach was used, since the solutions produced by the algorithms may be dependent on the movements developed for the initial vehicles released. If the batch-means approach were used the values of the later batches may have been affected by the initial batches, although there may not have been any direct correlations between them.

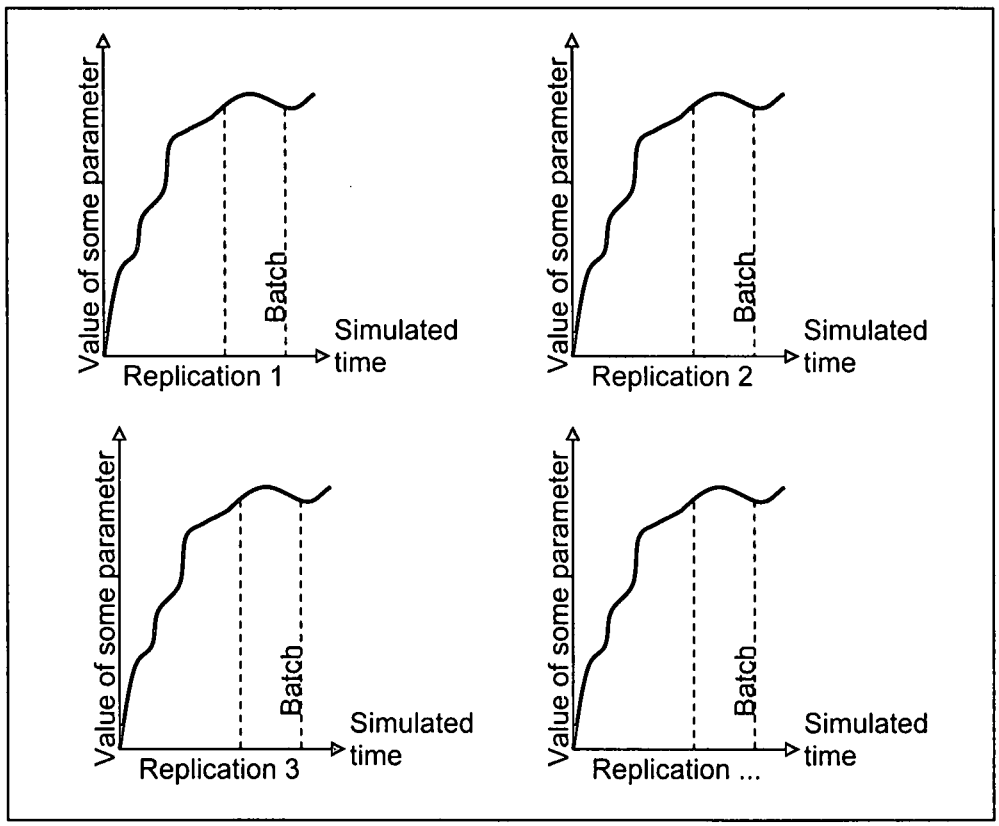



*Figure 6.2 Simulating the system as terminating*

The primary parameters of concern are the total cost parameters and service level as defined on page 76 in Section 5.3.1. (a) . The quality of the solutions for each scenario is of secondary concern. Consequently, the average late time of containers and the average transporter efficiency were also calculated.

Generally speaking, simulation models may require the calculation of warm-up and cool-down times. The warm-up period is the time required by the model to reach a steady state,

and the cool down period is a period with which the simulation run must be extended so that the period evaluated will remain in a steady state. In the cool-down state, containers are still entered into the system so that transporters will still be scheduled in the evaluation period for delivering these "future" containers.

The required period for which a model must be executed is thus the sum of the warm-up time, evaluation period and cool-down time. The following diagram (Figure 6.3) shows how the value of a parameter changes over the course of a simulation run.

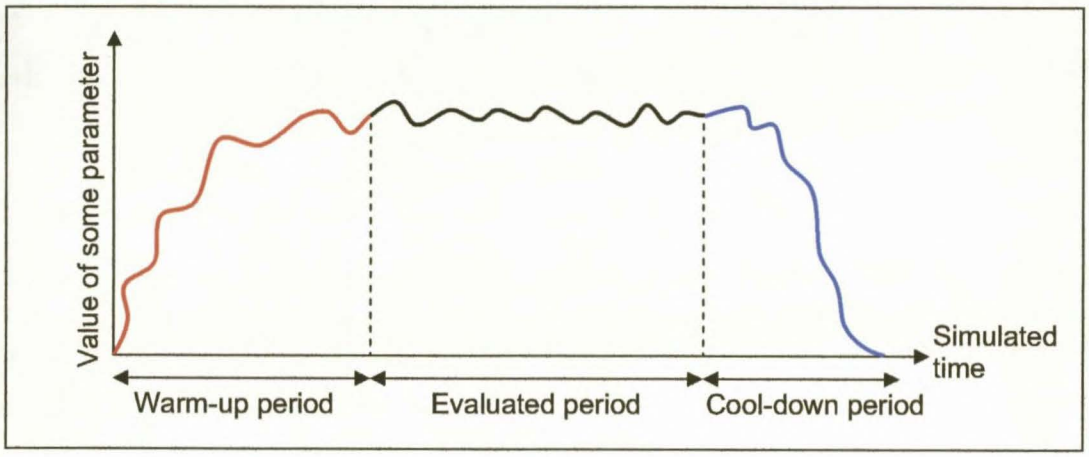



*Figure 6.3 Typical time values of a parameter during a simulation run*

If the state of the system from which a scenario is started represents the actual system state, no warm-up time should be considered. The scenarios evaluated started from an empty system and an appropriate warm-up time had to be determined.

A cool-down period is required to ensure that the values of the parameters are in steady state at the specific point in time when they are determined. In the Autocarriers simulation model vehicles, must be entered into the system beyond the evaluated period to ensure that carriers are requested and moved in the evaluated period (which affects the transportation cost in the evaluated period). A vehicle that is released beyond the evaluated period may result in a carrier moving in the evaluated period in order to reach the vertex at which the vehicle is released to avoid the late delivery of the vehicle.

Various replications were executed for each scenario to determine an appropriate warm-up time. The typical output obtained for a replication is illustrated in Figure 6.4.

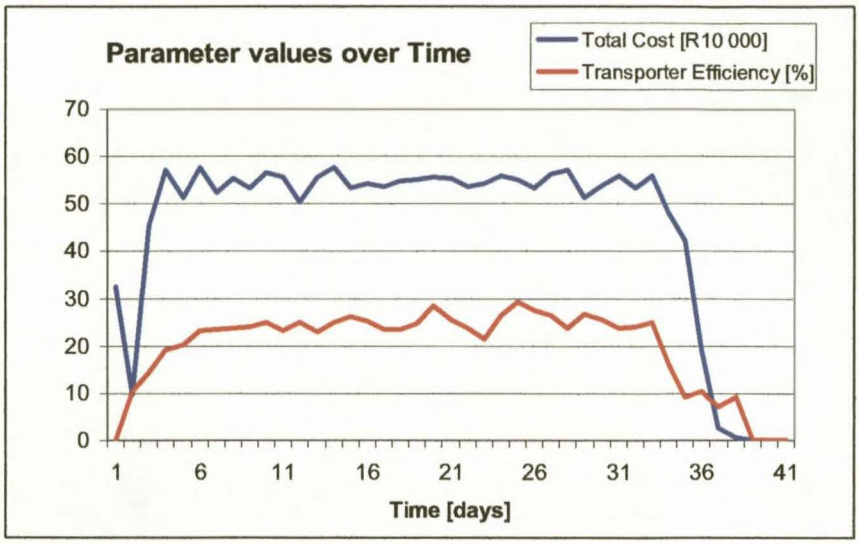



*Figure 6.4 Graph showing daily values of two parameters*

The graph illustrates the values of two parameters that were calculated on a daily basis. It can be seen on the graph that the system entered into a steady state after approximately 14 simulated days. The model was run for 31 simulated days. It is evident that the system enters the cool-down period after the 31st simulated day.

The simulation structure automatically enters containers into the system beyond the end of the evaluation period so that the system will remain in steady state until the end of the evaluation period.

After the warm-up time had been determined for a scenario, 10 runs were initially executed as a pilot study and the parameter values for these runs were calculated. The confidence interval for the mean total cost was computed, and the required number of runs estimated using a method outlined in Bekker [63] which states

$$n^* = n\left(\frac{h}{h^*}\right)^2, \qquad (6.1.\text{ A})$$

where $n^*$ is the estimated number of runs required, $n$ the number of runs in the pilot study, $h^*$ the desired confidence interval half-width for the parameter and $h$ the confidence interval half-width obtained for the parameter in the pilot study.

If the confidence interval half-width of the mean total cost was wider than 5% of the mean total cost (at a 95% confidence level), more runs were executed to narrow down the confidence interval.

## 6.2. SCENARIOS EVALUATED

Various scenarios were studied to illustrate the value of the model and algorithms developed. These scenarios are based on data received from Autocarriers. Assumptions were made for certain parameters where data was unavailable.

### 6.2.1. Default scenario

The default scenario used in this section models Autocarriers' transportation of vehicles among their Johannesburg, Cape Town, East London and Durban branches. The vehicle transportations among these branches account for the largest proportion of all vehicle transportations done by the company. Table 6.1 below contains the relevant data for each branch:

| Branch | Penalty rate of released containers | Average slack time allowed |
|---|---|---|
| Cape Town | 10 R/h | 96 h |
| Durban | 10 R/h | 72 h |
| East London | 20 R/h | 84 h |
| Johannesburg | 15 R/h | 108 h |

*Table 6.1 Default vertex information for Autocarriers problem*

Autocarriers classifies vehicles into two categories: high vehicles which can only be loaded onto the high slots on a carrier, and low vehicles that can be loaded onto any slot. The average historical delay time of a carrier is three hours. The following carrier configurations are available:

| Carrier | High slots | Low slots | Running cost | Average speed | Number available |
|---|---|---|---|---|---|
| Seven-Slot Carrier | 7 | 0 | R5.00/km | 55 km/h | 9 |
| Eight-Slot Carrier | 2 | 6 | R5.00/km | 50 km/h | 18 |
| Nine-Slot Carrier | 3 | 6 | R5.00/km | 55 km/h | 13 |
| Ten-Slot Carrier | 3 | 7 | R5.50/km | 55 km/h | 14 |
| Eleven-Slot Carrier | 4 | 7 | R5.50/km | 60 km/h | 22 |

*Table 6.2 Default fleet for Autocarriers problem*

The average number of vehicles moved (per month) among the various branches are listed in Table 6.3:

| Number of high vehicles transported [per month] | | | | | |
|---|---|---|---|---|---|
| **From \ To** | **Cape Town** | **Durban** | **EL** | **JHB** | **Total** |
| **Cape Town** | - | 8 | 9 | 46 | 63 |
| **Durban** | 592 | - | 310 | 500 | 1402 |
| **East London** | 164 | 97 | - | 319 | 580 |
| **Johannesburg** | 89 | 459 | 176 | - | 724 |
| **Total** | 845 | 564 | 495 | 865 | 2769 |
| **Number of low vehicles transported [per month]** | | | | | |
| **From \ To** | **Cape Town** | **Durban** | **EL** | **JHB** | **Total** |
| **Cape Town** | - | 134 | 30 | 315 | 476 |
| **Durban** | 396 | - | 431 | 2000 | 2827 |
| **East London** | 237 | 191 | - | 347 | 775 |
| **Johannesburg** | 396 | 640 | 248 | - | 1284 |
| **Total** | 1029 | 965 | 709 | 2662 | 5365 |

*Table 6.3 Average number of vehicles moved in the default scenario*

From the table above it can be seen that Johannesburg receives more vehicles than the other cities, while Durban sends off (or releases) the most vehicles.

Since the data describing the release times of the vehicles was not available it was assumed that the inter-arrival times (i.e. the inter-release times) of the vehicles are exponentially distributed and that the group size of released vehicles is one.

As stated before, the notification time is the time period between the time that Autocarriers have been notified that a vehicle must be moved and the time when it will become available for pickup. Longer notification times make better scheduling possible. A conservative value of 1.5 days has been chosen for the average notification time. (The actual notification times are usually in the order of a week.)

Autocarriers has not kept record of the notification time of the vehicles. Consequently, a theoretical distribution could not be determined. The Exponential distribution was used for the notification times. A short motivation for using this distribution can be stated as follows:

It is assumed that Autocarriers' knowledge of when vehicles will be released is a function that declines exponentially as depicted by Figure 6.5. The diagram depicts the probability of correct future knowledge possessed at point $T$. The probability of correct knowledge about events in the future declines the further away these events are in the future (relative to $T$).

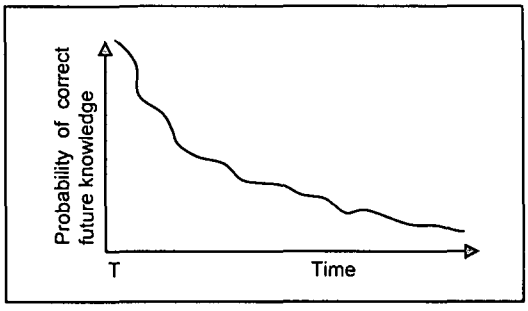



*Figure 6.5 Probability of correct future knowledge*

### 6.2.2. Comparing heuristics

Autocarriers' objective is the delivery of vehicles at a minimum transportation cost, without compromising their service level. The solution algorithms were benchmarked according to Autocarriers' objective function.

The results that were obtained are listed in Table 6.4. (The costs are expressed as costs per week.)

| Heuristic | Total penalty cost [R] | Total running cost [R] | Total cost [R] | Service level [%] | Ave. cont. late time [h] | Ave. transporter efficiency |
|---|---|---|---|---|---|---|
| 1. MSL | 0 | 3,094,498 | 3,094,498 | 100.00 | n/a | 35.36 |
| 2. MTC (δ=10) | 937,666 | 1,469,813 | 2,407,480 | 44.85 | 81.00 | 77.85 |
| 3. MTC (δ=15) | 884,754 | 1,479,504 | 2,364,258 | 41.17 | 78.49 | 78.44 |
| 4. MTC (δ=25) | 972,404 | 1,461,771 | 2,434,175 | 42.71 | 80.29 | 77.30 |
| 5. MTC (δ=150) | 513,520 | 1,683,677 | 2,197,197 | 52.31 | 52.04 | 73.41 |
| 6. MTC (δ=300) | 265,663 | 1,831,121 | 2,096,784 | 60.49 | 32.27 | 67.72 |
| 7. MTC (δ=600) | 179,071 | 1,900,834 | 2,079,904 | 70.03 | 29.97 | 64.87 |
| 8. MTC (δ=2000) | 118,417 | 1,871,580 | 1,989,997 | 84.51 | 15.49 | 64.39 |

*Table 6.4 Results obtained while comparing algorithms*

*Results concerning the MSL algorithm*: From the table above it can be seen that the maximum service level algorithm results in all containers being delivered on time. The transporter running cost in the MSL algorithm is much higher than that of the MTC

algorithms. This is to be expected, since the MSL algorithm ignores the running cost of the transporters when scheduling – only the delivery times of the containers are considered.

*Results concerning the MTC algorithm*: The MTC algorithm results in lower total costs, but it compromises the service level. A saving of roughly 20% to 35% is made when using this algorithm instead of the MLS algorithm, but a significant percentage of containers are delivered late (roughly 15% to 60%, depending on the value of $\delta$). If a container is delivered late, it will be delivered on average 0.5 – 3.5 days late. The transporter efficiency is almost doubled when using the minimum total cost algorithm, which is reflected in the transporter running cost being halved.

*Results concerning the $\delta$ parameter*: It is interesting to note the effect of the $\delta$ parameter on the results. The total penalty cost decreases as $\delta$ increases, while the total running cost increases. This is not surprising, as the parameter was designed to shift the focus from the transporter running cost to the container penalty cost.

There is a similar relationship between the transporter efficiency and the service level. As $\delta$ increases the service level increases, but the transporter efficiency is negatively affected. This can be explained as follows: In order to deliver containers on time, the transporters cannot wait for a prolonged period at a vertex. Consequently the transporters are not loaded to their full capacity before departing from a vertex.

The algorithm is not very sensitive to the $\delta$ value and it must be set to a high level to accomplish a good service level and total cost. This is due to the low penalty rates of the containers with regard to the transporter running costs. Take the example of a container that is transported from Cape Town to Johannesburg. The container has four days to be delivered to Johannesburg and a small penalty of R10 is paid for every hour that the container is late. The cost of the transporter travelling between these cities is at least R7 000. In order to ensure that the container is not delivered late, the $\delta$ value must scale the penalty cost to the same magnitude of order as the transporter cost. Consequently $\delta = 2\,000$ yields a better result than $\delta = 10$.

The optimum value of $\delta$ can be determined by finding the $\delta$ value that minimises the total cost. (Note that the total penalty cost and total running cost are played off against each other as $\delta$ changes.) Finding the optimum $\delta$ value for the Autocarriers problem requires a great number of simulations and was consequently not included in the scope of this project.

*Other aspects of the scenario:* An interesting aspect that can be examined is the late time distribution of the containers. The container late time distribution for a MTC ($\delta$ =10) solution is show in Figure 6.6 below:

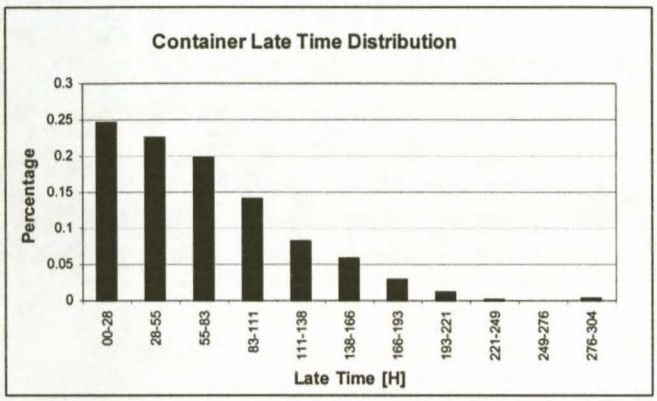



*Figure 6.6 Container late time distribution for MTC*

The distribution is skew to the left, which means that if a container is delivered late, it has a good chance of not being delayed for a prolonged period.

### 6.2.3. Decreasing the delay time

Autocarriers allows a three hour delay time for the transporters to load and unload vehicles. The effect of halving the delay time was examined to determine whether or not an effort should be made to reduce the loading and unloading time of carriers.

The following results were obtained when using the MTC($\delta$=10) algorithm (see Table 6.5). (The costs are expressed as costs per week.)

| Heuristic | Total penalty cost [R] | Total running cost [R] | Total cost [R] | Service level [%] | Ave. cont. late time [h] | Ave. transporter efficiency |
|---|---|---|---|---|---|---|
| 1. Delay Time = 3 h | 937,666 | 1,469,813 | 2,407,480 | 44.85 | 81.00 | 77.85 |
| 2. Delay Time = 1.5 h | 778,176 | 1,569,517 | 2,347,694 | 51.15 | 74.87 | 77.86 |

*Table 6.5 Results obtained when halving the delay time*

There is a small improvement in the total cost when the delay time is reduced. The confidence intervals of the total cost values should be narrowed further to confirm this improvement, since the current confidence ranges of these parameters overlap. An

estimated improvement of R60 000 per week may not justify reducing the delay time if it means buying expensive equipment and hiring extra personnel.

The small improvement in the total cost that results from decreasing the delay time can be explained as follows: the improvement is the result of higher transporter availabilities and faster container deliveries. Since there is a sufficient number of transporters available, the transporter availability does not place a limiting restriction on the system. A saving is made, however, since containers can be delivered faster (since the stop-over time in every town is reduced). Because the container penalty rates are low for the Autocarriers problem only a small saving is made.

Reducing the delay time will result in better savings in problems having the following features: many vertices (and thus stop-overs), a very restricted number of transporters (since the transporter availability is important in these cases) and high container penalty rates.

### 6.2.4. Changing the Carrier Fleet

An important aspect of the Autocarriers problem is to consider changes in the carrier fleet. Various changes in the fleet can be evaluated by the software tool, for example: changes to the carrier types in the problem, increasing the fleet size and decreasing the fleet size. The latter two changes were examined.

(a) Decreasing the fleet size

A question considered was whether a 100% service level can be maintained if the fleet size is reduced by 50%.

The following results (as listed in Table 6.6) were obtained for the weekly parameters when the MSL algorithm and MTC (δ=2000) algorithm were used:

| Heuristic | Total penalty cost [R] | Total running cost [R] | Total cost [R] | Service level [%] | Ave. cont. late time [h] | Ave. transporter efficiency |
|---|---|---|---|---|---|---|
| 1. MSL - Fleet 100% | 0 | 3,094,498 | 3,094,498 | 100.00 | n/a | 35.36 |
| 2. MSL - Fleet 50% | 0 | 1,789,594 | 1,789,594 | 100.00 | n/a | 61.00 |
| 3. MTC - Fleet 100% | 118,417 | 1,871,580 | 1,989,997 | 84.51 | 15.49 | 64.39 |
| 4. MTC - Fleet 50% | 472,550 | 1,500,708 | 1,973,258 | 49.84 | 50.16 | 77.82 |

*Table 6.6 Results obtained when decreasing the fleet size*

*Results concerning the MSL algorithm*: It is clear that a 100% service level can be maintained when the fleet size is decreased by 50%. It is interesting that the running costs are significantly reduced as a result of the decrease in the fleet size, since there are fewer carriers available. Fewer carriers can be requested unnecessarily by the algorithm and so the transporter efficiency increases. Evidence supporting this argument is that the ratio with which the transporter efficiency improves is very similar to the ratio with which the total running cost decreases.

*Results concerning the MTC algorithm:* When the MTC(δ=10) algorithm was applied to the same problem, an opposite effect was observed. It was found that the system would either not reach a steady state, or would reach a steady state with much higher costs. The MTC algorithm is reluctant to move the transporters due to high running costs, and consequently they were not moved often enough to deliver the containers on time.

Figure 6.7 illustrates the daily total cost that was retrieved from the solution of the MTC(δ=10) algorithm. It can be seen that the total cost parameter will converge to a steady state value of at least R400 000 per day.

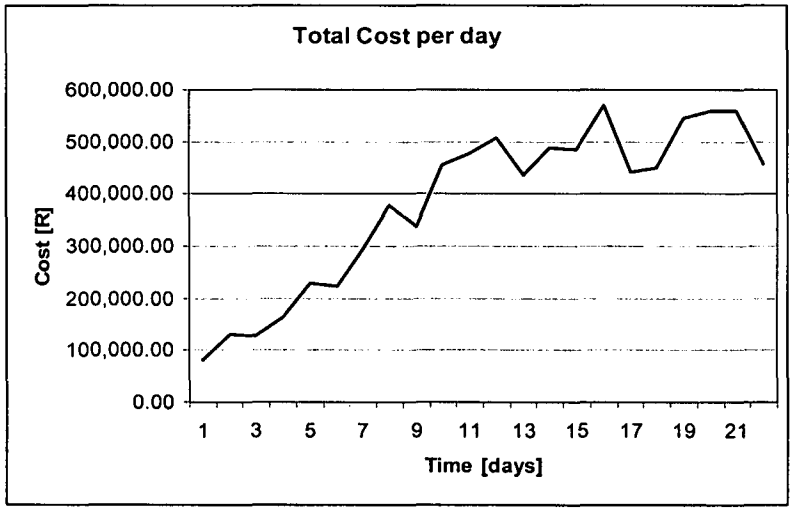



*Figure 6.7 The daily total cost of scheduling a 50% fleet with MTC(δ=10)*

The R400 000 per day total cost could be expressed as R2.8 million per week, which suggests that the δ value was too low. The MTC algorithm was reapplied to the problem, but with a much higher δ value of 2000.

It can be seen (in Table 6.6) that the total cost remains unaffected when the fleet size is decreased and scheduling is done with the MTC(δ=2000) algorithm. It is interesting to note that the total running cost decreased (due to fewer transporters travelling) and that the container penalty cost is increased (due to the lower transporter availability). It comes as no surprise that the service level is decreased and the average container late time is increased. This is the direct result of a smaller fleet. The smaller fleet is also used more effectively, since fewer transporters are available to move the same number of containers.

The MSL algorithm resulted in a slight lower cost than the MTC(δ=2000) algorithm. This suggests that the δ value is still not at an optimum for this problem configuration.

(b) Increasing the fleet size

The fleet size was increased by 25% and the influences on the MSL and MTC (δ=10) algorithms were examined.

The following results (listed in Table 6.7) were obtained for the weekly parameters:

| Heuristic | Total penalty cost [R] | Total running cost [R] | Total cost [R] | Service level [%] | Ave. cont. late time [h] | Ave. transporter efficiency |
|---|---|---|---|---|---|---|
| 1. MSL (Fleet 100%) | 0 | 3,094,498 | 3,094,498 | 100.00 | n/a | 35.36 |
| 2. MSL (Fleet 125%) | 0 | 4,367,787 | 4,367,787 | 100.00 | n/a | 24.96 |
| 3. MTC (Fleet 100%) | 937,666 | 1,469,813 | 2,407,480 | 44.85 | 81.00 | 77.85 |
| 4. MTC (Fleet 125%) | 840,101 | 1,536,839 | 2,376,940 | 51.49 | 48.51 | 78.58 |

*Table 6.7 Results obtained when increasing the fleet size*

*Results concerning the MSL algorithm*: The MSL algorithm did not yield desirable results when increasing the fleet size, since there were more transporters which the algorithm could move unnecessarily. The results shown above confirm that the transporter running cost increases, while the transporter efficiency decreases when a larger fleet is available.

*Results concerning the MTC algorithm:* The MTC algorithm was not greatly affected by the increase of the fleet size, since enough carriers were available in the original fleet. The MTC algorithm did not create more movements just because more carriers were available. A slight decrease in the container penalty cost was evident, as well as improvements in the average container late time and service level. The reason why these parameters improved is that more transporters were available in the system, thus the probability that a transporter would be stationed at a vertex when a container was released, was increased. The algorithm would

be keener to create a movement for the container, because a transporter with a low transit cost was available (since the transporter did not have to be moved from another vertex to the current vertex of the container). As a consequence more direct movements were created and the aforementioned parameters were improved. The total running cost was slightly increased as a result of the larger number of direct movements.

### 6.2.5. Adding a depot

The influence of adding a depot to the system was examined. A depot in Bloemfontein would allow carriers to run to and from the depot, while exchanging cargo with each other. A carrier could, for example, pick up two vehicles in Johannesburg destined for Cape Town and Durban respectively and take them to Bloemfontein. The vehicle destined for Durban could then be delivered at the depot while the carrier travelled to Cape Town. The vehicle destined for Durban could then be transported by another carrier that was routed via Bloemfontein to Durban.

The following results (as listed in Table 6.8) were obtained for the weekly parameters when using the MLS and MTC(δ=600) algorithms:

| Heuristic | Total penalty cost [R] | Total running cost [R] | Total cost [R] | Service level [%] | Ave. cont. late time [h] | Ave. transporter efficiency |
|---|---|---|---|---|---|---|
| 1. MSL without depot | 0 | 3,094,498 | 3,094,498 | 100.00 | n/a | 35.36 |
| 2. MSL with depot | 0 | 2,433,655 | 2,433,655 | 100.00 | n/a | 39.52 |
| 3. MTC without depot | 179,071 | 1,900,834 | 2,079,904 | 70.03 | 29.97 | 64.87 |
| 4. MTC with depot | 113,530 | 1,985,162 | 2,098,692 | 82.38 | 17.62 | 60.21 |

*Table 6.8 Results obtained when adding a depot*

*Results concerning the MSL algorithm*: When the MSL scheduling algorithm is used, a significant saving can be made by adding a depot. The saving is the result of higher transporter efficiency.

*Results concerning the MTC algorithm:* The total cost remains almost unaffected when a depot is added if scheduling is done with the MTC algorithm. The total penalty cost decreased by roughly the same amount by which the total running cost increased. The higher total running cost was the result of lower transporter availability due to the extra stopover. Adding a depot results in a lower penalty rate and higher service level, as well as an improved average late time. If these aspects are of great concern, a depot can be added, but the depot will not result in a lower total cost.

### 6.2.6. Adding a customer

Autocarriers often has to decide whether or not to tender for a new contract. Part of the tender decision is to determine the minimum tender offer. The minimum tender offer is the income required from a contract to break even on a deal.

It is difficult to determine the break even point, since it is dependent on various factors: the type of vehicles that are involved in the deal; the distribution of vehicle inter-arrival times; the notification time offered by the client and the configuration of other clients. It may be profitable to accept a deal for moving cars from Cape Town to Johannesburg but not vice versa, since the carriers in the original configuration may be running empty from Cape Town to Johannesburg.

A scenario investigated was determining the minimum tender offer if Autocarriers could choose to bargain on moving 2 000 vehicles per month from Durban to Cape Town. The vehicle arrivals are Poisson distributed, and have a notification time of six hours. There is a 50-50 split between high vehicles and low vehicles. This was an interesting scenario to investigate, since many vehicles are being transported from Durban to Cape Town, but few are transported in the opposite direction. The effect of this tender is that transporters transport full loads down to Cape Town, but return empty.

The following results (as listed in Table 6.9) were obtained for the weekly parameters when using the MLS and MTC(δ=600) algorithms:

| Heuristic | Total penalty cost [R] | Total running cost [R] | Total cost [R] | Service level [%] | Ave. cont. late time [h] | Ave. transporter efficiency |
|---|---|---|---|---|---|---|
| 1. MSL without tender | 0 | 3,094,498 | 3,094,498 | 100.00 | n/a | 35.36 |
| 2. MSL with tender | 9,225 | 3,297,527 | 3,306,753 | 95.94 | 4.06 | 47.93 |
| 3. MTC without tender | 179,071 | 1,900,834 | 2,079,904 | 70.03 | 29.97 | 64.87 |
| 4. MTC with tender | 208,092 | 3,380,068 | 3,588,160 | 74.22 | 25.79 | 53.51 |

*Table 6.9 Results obtained when adding a customer*

An important finding is that if the tender is accepted, a 100% service level cannot be maintained. From the table above it can be seen that the service level drops with 4% when the MSL algorithm is used. The impact of the tender on the two algorithms is discussed below.

*Results concerning the MSL algorithm:* Since a 100% service level could not be maintained, a penalty cost was incurred. This penalty is small, because the average late time of containers is only four hours, and only 4% of the containers are delivered late. The transporter efficiency increased, since more containers were transported with the same number of transporters and so the transporters were more efficiently loaded.

If the increase in the weekly total cost is expressed as a cost per vehicle transported, the minimum tender offer is R460 per vehicle, when using the MSL algorithm.

*Results concerning the MTC algorithm:* As with the MSL algorithm, the total penalty and total running costs increased. The increase in the total running cost of the MTC algorithm was much higher than that of the MSL algorithm, although the total costs in both cases were roughly the same if the tender was accepted. The MTC algorithm delivered a higher total cost than the MSL algorithm suggesting that the $\delta$ parameter should be adjusted to find a lower total cost with the MTC algorithm.

The larger increase in the total cost when the MTC algorithm was used is an indication that it is a good algorithm to minimise the total cost in a problem where there is balance in the flow of vehicles between cities. If the tender was accepted, there were many vehicles going to Cape Town but few leaving, and consequently, there was not much that could be done to minimise the total cost.

The transporter efficiency was negatively affected in the MTC case, since the original transporter efficiency was very good. When the carriers returned from Cape Town they were empty and so the transporter efficiency decreased. It is interesting to note that with the MSL algorithm the transporter efficiency increased, since it had a low initial value. The great number of containers moving in one direction improved the transporter efficiency, in spite of the transporters returning empty. The MTC algorithm started with an initially high value, which was then negatively effected by the transporters returning empty.

There was a small improvement in the service level and average container late time when the MTC algorithm was used. This improvement was due to the high flow of containers from Durban to Cape Town. The high flow increases the probability that a transporter will be travelling on the same movement as that required by a newly-released container. It should be noted that the improved container late time does not imply that the containers that are part of the original setup will be delivered earlier. A small average late time was experienced by the new containers and this decreased the average late time. (A small average late time

was experienced by the new containers, because the chance that a container could be routed to its destination, Cape Town, without being late, was high due to the large number of trucks moving in that direction).

When using the MTC ($\delta$ = 600) algorithm it was found that the minimum tender offer was much higher than that of the MSL algorithm. If the tender offer is expressed as a cost per vehicle transported, the MTC algorithm suggests Autocarriers should not bargain for less than R3 270 per vehicle. This amount is much higher than that suggested by the MSL algorithm. This is because of the huge impact the tender will have on the increase in the total cost of the system in the MTC case (R1.5m) compared to the MSL case (R0.2m).

The software tool can be used to examine the effect of a tender on each branch (or city). For example, the change in the container late time and service level can be calculated for a specific city. Such an in-depth analysis is required in practice, but was deemed unnecessary in the scope of this thesis.

The conclusion of the case scenarios is discussed in the following section.

## 6.3. CHAPTER SUMMARY

This chapter contains a discussion on how the scheduling model (of chapter 3) and solution algorithms (described in chapter 4) in the software tool (discussed in chapter 5) can be used to address the Autocarriers problem. Solutions to the Autocarriers problem were simulated and evaluated according to their strategic business objectives.

The MSL algorithm is an effective algorithm for the Autocarriers problem, since it reflects their primary business objective, which is to maximise their service level. A significant saving can, however, be achieved by using the MTC algorithm. The MTC algorithm uses the transporters more efficiently, but consequently reduces the service level. As the $\delta$ parameter value is increased in the MTC algorithm, the total penalty cost is reduced, while the total running cost increases. Therefore, the $\delta$ parameter is useful for shifting the emphasis between these costs.

When the MTC algorithm is used for scheduling the transporters, no significant improvement can be seen when decreasing the delay time. This is because Autocarriers has a low penalty cost, a large transporter fleet and a small number of vertices.

When the fleet size is reduced, the MSL algorithm yields a solution with a lower total cost, since fewer transporters are available that can be moved unnecessarily by the algorithm. Likewise, if the fleet size is increased, the MSL algorithm yields a solution of higher cost since more transporters can be moved while carrying only a few containers. The total cost of the MTC algorithm is not greatly affected by a change in the fleet size, but a higher $\delta$ value should be chosen when scheduling a small fleet.

A central depot facility was added to the problem and the effect of the new depot examined. The depot reduced the total running cost of the MSL algorithm without affecting the 100% service level, as was expected. When the MTC algorithm was applied to the problem, it was found that no significant saving could be achieved, although the service level and average container late time were improved.

The minimum tender value for a new contract could be determined with the simulation model, as was demonstrated. A tender for moving a large, unbalanced number of vehicles was examined. In the examined case it was found that the total cost of the MSL algorithm was less affected by the tender than the total cost of the MTC algorithm. Consequently, the minimum tender offer is much higher in the MTC algorithm. If the tender is accepted, the total costs of the two algorithms are very similar, indicating that the MTC algorithm is an effective algorithm to reduce the total cost when such a saving can be made. In an unbalanced problem it is more difficult to reduce the total cost. This explains the similarities between the total costs obtained by the two algorithms if the tender is accepted.

The conclusions of the project are discussed in the next chapter.

# Chapter 7 Conclusions

The conclusions of the thesis are described in this chapter. The conclusions regarding the scheduling model and solution algorithms are discussed first, after which the conclusions regarding the software tool are discussed.

## 7.1. THE GENERIC SCHEDULING MODEL AND SOLUTION ALGORITHMS

The generic problem definition consisting of transporters, containers, loading rules, arcs, vertices and movements can describe a variety of real-world problems, including the Autocarriers problem.

The scheduling model developed is simple but versatile and can be used to solve instances of the generic problem. The important aspects of a solution have been defined and can be used in the objective function of the model. The objective function is not limited to these parameters and a unique objective function can be specified. The two aspects that were focused on were the service level and total cost of the solution.

Two algorithms were developed to optimise these aspects. The maximum service level (MSL) algorithm is a very efficient algorithm, since it is able to maintain a high service level in a solution even though there are severe constraints on the number of transporters available. The MSL algorithm functions well under these restrictions. If the restriction on the number of transporters is eased, the algorithm may become less desirable as it results in high total running costs (although the penalty cost may be decreased). If more transporters are available in the problem, the MSL algorithm will use these transporters. Consequently, the transporters may not be loaded with many containers and this yields a high total running cost.

The minimum total cost (MTC) algorithm attempts to balance the total running cost with the total penalty cost. The algorithm has a parameter that can be set to shift the emphasis between these costs. Generally speaking, the MTC algorithm will result in a lower total cost, although the service level will decrease. The results of the MTC algorithm are not greatly affected when the fleet size is changed, although the optimum $\delta$ value may be larger for a small fleet.

When a problem is very unbalanced (i.e. when a large number of containers are moved in one direction and a small number in the opposite direction), it is more difficult to minimise the total cost of a solution. Consequently the MTC algorithm results in solutions with higher total costs, which are comparable with the total costs of solutions generated with the MSL algorithm.

## 7.2. SOFTWARE TOOL

The software tool is an adequate tool for defining an instance of the generic problem. Problems of realistic size can be defined with the structures of the software tool. The container templates and transporter templates ease the definition of the transporters and containers, since certain attributes can be derived from the templates and do not have to be entered repeatedly. The process of adding containers is still time-consuming, since a real-world problem may have hundreds of containers.

Solutions can easily be developed with the solution development structure. The manual scheduling structure is useful for ensuring that a valid solution is created. It is, however, difficult to understand the overall effect on the system when a new movement is created and when containers are assigned to this movement. The automatic scheduling function is thus useful for scheduling a large number of transporters and containers.

The solution evaluation structure is helpful in evaluating a solution. The structure is also flexible enough that various parameters (including user-defined parameters) can be calculated for a solution.

The simulation capability of the software tool enables the user to examine various changes to a problem. The simulation models are easy to set up and can be used to describe a large variety of scenarios. However, the run-time of the models is long. This is due to the slow connection between the program structures and the database. In practice, quick results are seldom required for this problem type and a client can usually wait a day or two for results.

The data analyser is a useful program feature, since a variety of statistical analyses are simplified. The user can easily plot a theoretical distribution on a sample histogram. The user can also request a chi-squared test on the proposed distribution.

The recommendations resulting from the project are stated in the next chapter.

# Chapter 8 Recommendations

This chapter lists the recommendations that could improve the scheduling model and the software tool.

## 8.1. THE SCHEDULING MODEL AND SOLUTION ALGORITHMS

Although the problem definition of the generic problem is useful for describing various problem instances, specific attention was given to the Autocarriers problem. A few changes to the problem definition can make the problem even more universal.

Firstly, the cargo capacity restriction which is currently defined as a set of loading rules can be replaced by a set of continuous functions. Consequently, not only an integer number of containers can then be loaded on a transporter, but any fractional value. The definition of a container entity will also require revision.

Secondly, restrictions can be added so that not all transporters are allowed to travel on all arcs. This is useful for restricting transporters to specific arcs and/or hindering them from moving on other arcs. Transporters of various types (e.g. trucks, ships and aeroplanes) can thus be incorporated in the same model. More than one arc must then also be allowed between two vertices in the same direction.

As mentioned above, the Autocarriers problem was the primary problem for which the model and algorithms were developed. The two solution algorithms are designed to develop a solution that optimises their strategic business objectives. More algorithms can be developed so that a wider range of objective functions can be optimised.

A specific aspect that may be addressed in the development of better solution algorithms is the minimisation of reloading containers from one carrier to another, since this may result in costs due to damage to containers, and longer time delays at depots.

The possibility of improving the initial solution can also be examined. The author believes that it is possible that a post-optimisation algorithm (or route improvement algorithm) can be developed for this problem type, although this was not done in the scope of this project.

## 8.2. THE SOFTWARE TOOL

The software tool that was developed is a very user-friendly and versatile tool. A few changes should however be made to enable a full-scale implementation of the tool at a company like Autocarriers.

The most important feature that should be added is an adapter than can retrieve information from another database and import it into the software tool's database. This is especially required for defining containers, since there can be hundreds of containers that are part of a problem definition.

The chi-squared test in the software tool is currently limited to the Normal and Exponential distributions and can be expanded to the other distributions. This is relatively simple to do, but due to time constraints this was not completed by the end of this project.

It is recommended that the simulation structure be enhanced. The specific area of concern is the run-time of large simulation models. The run-time can be reduced by loading the problem definition into the RAM and addressing it directly through variables. Currently the problem definition is kept in the database structure addressed through SQL queries. Addressing the problem through SQL queries results in large overheads.

# Chapter 9 Project Summary

A generic problem was defined to describe a variety of problem instances. The generic problem was defined using the following entities: transporters, containers, loading rules, arcs, vertices and movements. Similar transporters and containers can be grouped in template families to simplify the definition of these entities for a problem instance. One of the problems that could be described with the model was the Autocarriers problem.

The generic problem definition was used as a basis to develop a scheduling model. The model consists of an objective function and solution space. The objective function can either be defined from a combination of standard parameters or from user-defined parameters. The solution space of the problem is the set of all solutions that comply with the entity conditions described.

A software tool was developed that can facilitate the creation of instances of the generic problem and solutions to the problem. The definition of a problem instance and its solution are stored in a database structure so that they can be duplicated and easily configured.

Solutions to the problem can either be created by the user (aided by the software tool) or by the software tool itself. Two heuristic algorithms are available in the software, namely the maximum service level heuristic and the minimum total cost heuristic. These were developed to address two important aspects of the Autocarriers problem, which are to maximise the service level of a solution and to minimise the total cost of a solution respectively. After a solution has been created, the software tool can evaluate the solution according to the objective function of the scheduling model.

The software tool was given a simulation capability so that various changes to a problem instance can be examined and to determine which algorithm delivers the most desirable results. Changes to a problem instance that can be examined include the following: adding a customer, changing the transporter fleet, changing the transportation network, adding or removing a depot facility and changing the delay time of the transporters.

A data analyser was added to the software tool so that representative simulation models can be created and so that the simulation output can be examined in depth.

# References

[1] Dantzig GB, Ramser JH. The truck dispatching problem. Mngt Sci 1959;6:80-91.

[2] Toth P, Vigo D. The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics; 2002. pp. 1-26.

[3] Achuthan NR, Caccetta L, Hill SP. On the vehicle routing problem. Nonlinear Analysis, Theory, Methods and Applications 1997;30(7):4277-88.

[4] Toth P, Vigo D. The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics; 2002. pp. 157-93.

[5] Giosa ID, Tansini IL, Viera IO. New assignment algorithms for the multi-depot vehicle routing problem. J Opl Res Soc 2002;53:977-84.

[6] Lau HC, Sim M, Teo KM. Vehicle routing problem with time windows and a limited number of vehicles. Eur J Opl Res 2003;148:559-69.

[7] Renaud J, Boctor FF. A sweep-based algorithm for the fleet size and mix vehicle routing problem. Eur J Opl Res 2002;140:618-28.

[8] Gendreau M, Laporte G, Musaraganyi C, Taillard ED. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. Computers Opns Res 1999;26:1153-73.

[9] Dullaert W, Janssens GK, Sörensen K, Vernimmen B. New heuristics for the Fleet Size and Mix Vehicle Routing Problem with Time Windows. J Opl Res Soc 2002;53:1232-8.

[10] Golden B, Assad A, Levy L, Gheysens F. The Fleet Size and Mix Vehicle Routing Problem. Computers Opns Res 1984;11:49-66.

[11] Tarantilis CD, Kiranoudis CT, Vassiliadis VS. A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. Eur J Opl Res 2004;152:148-58.

[12] Toth P, Vigo D. The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics; 2002. pp. 225-243.

[13] Toth P, Vigo D. An exact algorithm for the vehicle routing problem with backhauls. Transport Sci 1997;31:372-85.

[14] Dror M, Trudeau P. Savings by split delivery routing. Transport Sci 1989;23(2):141-9.

[15] Bodin L, Mingozzi A, Baldacci R, Ball M. The Rollon-Rolloff Vehicle Routing Problem. Transport Sci 2000;34(3):271-88.

[16] Toth P, Vigo D. The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics; 2002. pp. 331-53.

[17] Sariklis D, Powell S. A heuristic method for the open vehicle routing problem. J Opl Res Soc 2000;51:564-73.

[18] Laporte G, Gendreau M, Potvin J-Y, Semet F. Classical and modern heuristics for the vehicle routing problem. Intl Trans Op Res 2000;7:285-300.

[19] Cordeau J-F, Gendreau M, Laporte G, Potvin J-Y, Semet F. A guide to vehicle routing heuristics. J Opl Res Soc 2002;53:512-22.

[20] Lenstra JK, Rinnooy Kan AHG. Complexity of vehicle routing and scheduling problems. Networks 1981;11:221-7.

[21] Garvin WM, Crandall HW, John JB, Spellman RA. Applications of linear programming in the oil industry. Mgmt Sci 1957;3:407-30.

[22] Balinski ML, Quandt RE. On an integer program for a delivery problem, Opns Res 1964;12(2):300-4.

[23] Laporte G, Mecure H, Nobert Y. An exact algorithm for the asymmetrical capacitated vehicle routing problem. Networks 1986;16:33-46.

[24] Fischetti M, Toth P, Vigo D. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. Opns Res 1994;42:846-859.

[25] Fischetti M, Toth P. An additive bounding procedure for combinatorial optimization. Opns Res 1989;37:319-328.

[26] Fisher ML, Optimal solution of vehicle routing problems using minimum k-trees. Opns Res 1994; 42:626-42.

[27] Miller DL. A matching based exact algorithm for capacitated vehicle routing problems. ORSA Journal on Computing 1995;7:1-9.

[28] Hadjiconstantinou E, Christofides N, Mingozzi A. A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations. Ann Oper Res 1995;61:21-43.

[29] Christofides N, Mingozzi A, Toth P. Exact algorithms for the vehicle routing problem based on the spanning tree and shortest path relaxations. Math Program 1981;20:255-82.

[30] Augerat P, Belenguer JM, Benavent E, Corberàn A, Naddef D, Rinaldi G. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report RR 949-M. Grenoble, France: Université Joseph Fourier; 1995.

[31] Lysgaard J, Letchford AN, Eglese RW. A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math Program 2004;Ser A 100:423-45.

[32] Desrochers M, Desrosiers J, Solomon M. A new optimization algorithm for the vehicle routing problem with time windows. Opns Res 1992;40(2):342-354.

[33] Wren A, Holliday A. Computer scheduling of vehicles from one or more depots to a number of delivery points. Operational Research Quarterly 1971;23:333-44.

[34] Gillett BE, Miller LR. A heuristic algorithm for the vehicle dispatch problem. Opns Res 1974;22:340-49.

[35] Toth P, Vigo D. The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics; 2002. p. 120.

[36] Clarke G, Wright JV. Scheduling of vehicles from a central depot to a number of delivery points. Opns Res 1964;12:568-581.

[37] Desrochers M, Verhoog TW. A matching based savings algorithm for the vehicle routing problem. Technical Report Cahiers du GERAD G-89-04, École des Hautes Études Commerciales de Montréal, Canada, 1989.

[38] Fisher ML, Jaikumar R. A generalized assignment heuristic for vehicle routing. Networks 1981;11:109-124.

[39] Lin S. Computer solutions of the travelling salesman problem. Bell System Technical Journal 1965;44:2245-69.

[40] Or I. Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking. PH.D. dissertation, Department of Industrial Engineering and Management Science, Northwestern University; Evanston, IL; 1976.

[41] Lin S, Kernighan BW. An effective heuristic algorithm for the travelling salesman problem. Opns Res 1973;21:498-516.

[42] Van Breedam A. An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints. Ph.D. dissertation. University of Antwerp; 1994.

[43] Bachem A, Hochstättler W, Malich M. The simulated trading heuristic for solving vehicle routing problems. Discrete Applied Mathematics 1996;65:47-72.

[44] Dorigo M. Optimization, learning and natural algorithms. Doctoral Dissertation. Polotecnico di Milano; Italy; 1992.

[45] Bullnheimer B, Hartl RF, Strauss C. Applying the Ant System to the Vehicle Routing Problem. Proceedings of the 2nd International Conference on Metaheuristics. Sophia-Antipolis, France; July 1997.

[46] Bullnheimer B, Hartl RF, Strauss C. An improved Ant System algorithm for the Vehicle Routing Problem. Ann Oper Res 1999;89:319-28.

[47] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1997;1:53-66.

[48] Bell JE, McMullen PR. Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics 2004;18:41-8.

[49] Holland JH. Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press; 1975.

[50] Baker BM, Ayechew MA. A genetic algorithm for the vehicle routing problem. Computers Opns Res 2003;30:787-800.

[51] Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers Opns Res 2004;31:1985-2002.

[52] Taillard ED. Parallel iterative search methods for vehicle routing problems. Networks 1993;23:661-673.

[53] Xu J, Kelly JP. A network flow-based tabu search heuristic for the vehicle routing problem. Transport Sci 1996;30:379-93.

[54] Toth P, Vigo D. The granular search (and its application to the vehicle routing problem). Technical report OR/98/9, DEIS, Università di Bologna: Italy.

[55] Cordeau J-F, Laporte G and Mercier A. A unified tabu search heuristic for vehicle routing problems with time windows. J Opl Res Soc 2001;52:928-36.

[56] Rochat Y, Taillard ED. Probabilistic diversification and intensification in local search for vehicle routing. J Heuristics 1995;1:147-67.

[57] Fogel DB, Michalewicz Z. How to solve it: modern heuristics. Berlin: Springer; 2002. pp. 117-125

[58] Agbegha GY, Ballou RH, Mathur K. Optimizing auto-carrier loading. Transport Sci 1998;32(2):174-88.

[59] Ruiz R, Maroto C, Alcaraz J. A decision support system for a real vehicle routing problem. Eur J Opl Res 2004;153:593-606.

[60] Desrochers M, Jones CV, Lenstra JK, Savelsbergh MW, Stougie L. Towards a model and algorithm management system for vehicle routing and scheduling problems. Decision Support Systems 1999;25:109-33.

[61] Davies AN. The development and use of routing functionality within Spoornet, Department of Industrial Engineering [unpublished thesis]. University of Stellenbosch, February 1998.

[62] Nemhauser GL, Rinnooy Kan AHG, Todd MJ. Handbooks in Operations Research and Management Science - Volume 1 Optimization. Amsterdam: North-Holland; 1989. 263-5.

[63] Bekker J. Simulation of discrete-event stochastic processes. Unpublished course notes. Stellenbosch: University of Stellenbosch; 2003. p 47.

# Bibliography

## I. BOOKS

1. Devore JL, Farnum NR. Applied Statistics for Engineers and Scientists. International: Duxbury, 1999.
2. Dumitrescu D, Lazzerini B, Jain LC, Dumitrescu A. Evolutionary computation. United States of America: CRC Press; 2002.
3. Finney RL, Thomas GB, Weir MD. Calculus, second edition. International: Addison-Wesley; 1994.
4. Fogel DB, Michalewicz Z. How to solve it: modern heuristics. Berlin: Springer; 2002.
5. Glover F, Kochenberger GA. Handbook of metaheuristics. Boston: Kluwer Academic Publishers; 2003.
6. Hastings NAJ, Peacock JB. Statistical Distributions – A handbook for students and practitioners. London: Butterworths; 1975.
7. Kendall KE, Kendall JE. Systems analysis and design, fifth edition. United States of America: Prentice Hall; 2002.
8. Law AM, Kelton WD. Simulation Modeling and Analysis, thrid edition. International: McGraw Hill; 2000.
9. Nemhauser GL, Rinnooy Kan AHG, Todd MJ. Handbooks in operations research and management science - Volume 1 optimization. Amsterdam: North-Holland; 1989.
10. Toth P, Vigo D. The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics; 2002.
11. Winston WL. Operations research - applications and algorithms. International: Thompson; 2004.

## II. ARTICLES

12. Achuthan NR, Caccetta L, Hill SP. On the vehicle routing problem. Nonlinear Analysis, Theory, Methods and Applications 1997;30(7):4277-88.
13. Agbegha GY, Ballou RH, Mathur K. Optimizing auto-carrier loading. Transport Sci 1998;32(2):174-88.
14. Arunapuram S, Mathur K, Solow D. Vehicle routing and scheduling with full truckloads. Transport Sci 2003;37(2):170-82.
15. Bachem A, Hochstättler W, Malich M. The simulated trading heuristic for solving vehicle routing problems. Discrete Applied Mathematics 1996;65:47-72.
16. Baker BM, Ayechew MA. A genetic algorithm for the vehicle routing problem. Computers Opns Res 2003;30:787-800.

17. Bell JE, McMullen PR. Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics 2004;18:41-8.
18. Berger J, Barkaoui M. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. Computers Opns Res 2004;31:2037-53.
19. Bodin L, Mingozzi A, Baldacci R, Ball M. The Rollon-Rolloff Vehicle Routing Problem. Transport Sci 2000;34(3):271-88.
20. Bullnheimer B, Hartl RF, Strauss C. An improved Ant System algorithm for the Vehicle Routing Problem. Ann Oper Res 1999;89:319-28.
21. Bullnheimer B, Hartl RF, Strauss C. Applying the Ant System to the Vehicle Routing Problem. Proceedings of the 2nd International Conference on Metaheuristics. Sophia-Antipolis, France; July 1997.
22. Clarke G, Wright JV. Scheduling of vehicles from a central depot to a number of delivery points. Opns Res 1964;12:568-581.
23. Cordeau J-F, Gendreau M, Laporte G, Potvin J-Y, Semet F. A guide to vehicle routing heuristics. J Opl Res Soc 2002;53:512-22.
24. Chu C-W. A heuristic algorithm for the truckload and less-than-truckload problem. Eur J Opl Res 2004.
25. Dantzig GB, Ramser JH. The truck dispatching problem. Mngt Sci 1959;6:80-91.
26. Desrochers M, Desrosiers J, Solomon M. A new optimization algorithm for the vehicle routing problem with time windows. Opns Res 1992;40(2):342-354.
27. Desrochers M, Jones CV, Lenstra JK, Savelsbergh MW, Stougie L. Towards a model and algorithm management system for vehicle routing and scheduling problems. Decision Support Systems 1999;25:109-33.
28. Dullaert W, Janssens GK, Sörensen K, Vernimmen B. New heuristics for the Fleet Size and Mix Vehicle Routing Problem with Time Windows. J Opl Res Soc 2002;53:1232-8.
29. Fischetti M, Toth P, Vigo D. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. Oper Res 1994;42(5):846-59.
30. Fischetti M, Toth P. An additive bounding procedure for combinatorial optimization. Opns Res 1989;37:319-328.
31. Gendreau M, Laporte G, Musaraganyi C, Taillard ED. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. Computers Opns Res 1999;26:1153-73.
32. Gillett BE, Miller LR. A heuristic algorithm for the vehicle dispatch problem. Opns Res 1974;22:340-49.
33. Giosa ID, Tansini IL, Viera IO. New assignment algorithms for the multi-depot vehicle routing problem. J Opl Res Soc 2002;53:977-84.
34. Golden B, Assad A, Levy L, Gheysens F. The Fleet Size and Mix Vehicle Routing Problem. Computers Opns Res 1984;11:49-66.

35. Hassin R, Rubinstein S. On the complexity of the *k*-customer vehicle routing problem. Opns Res Lett 2005;33:71-6.
36. Ho SC, Haugland D. A tabu search heuristic for the vehicle routing problem with time windows and split deliveries. Computers Opns Res 2004;31:1947-64.
37. Ioannou G, Kritikos M, Prastacos G. A greedy look-ahead heuristic for the vehicle routing problem with time windows. J Opl Res Soc 2001;52:523-37.
38. Kim J-U, Kim Y-D, Shim S-O. Heuristic algorithms for a multi-period multi-stop transportation planning problem. J Opl Res Soc 2002;53:1027-37.
39. Laporte G, Gendreau M, Potvin J-Y, Semet F. Classical and modern heuristics for the vehicle routing problem. Intl Trans Op Res 2000;7:285-300.
40. Laporte G, Mecure H, Nobert Y. An exact algorithm for the asymmetrical capacitated vehicle routing problem. Networks 1986;16:33-46.
41. Lau HC, Sim M, Teo KM. Vehicle routing problem with time windows and a limited number of vehicles. Eur J Opl Res 2003;148:559-69.
42. Lenstra JK, Rinnooy Kan AHG. Complexity of vehicle routing and scheduling problems. Networks 1981;11:221-7.
43. Lysgaard J, Letchford AN, Eglese RW. A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math Program 2004;Ser A 100:423-45.
44. Miller DL. A matching based exact algorithm for capacitated vehicle routing problems. ORSA Journal on Computing 1995;7:1-9.
45. Nagy G, Salhi S. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. Eur J Opl Res 2005;162:126-41.
46. Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers Opns Res 2004;31:1985-2002.
47. Poot A, Kant G, Wagelmans AP. A savings based method for real-life vehicle routing problems. J Opl Res Soc 2002;53:57-68.
48. Ralphs TK, Kopman L, Pulleyblank WR, Trotter LE. On the capacitated vehicle routing problem. Math Program 2003;Ser B 94:343-59.
49. Renaud J, Boctor FF. A sweep-based algorithm for the fleet size and mix vehicle routing problem. Eur J Opl Res 2002;140:618-28.
50. Ruiz R, Maroto C, Alcaraz J. A decision support system for a real vehicle routing problem. Eur J Opl Res 2004;153:593-606.
51. Sariklis D, Powell S. A heuristic method for the open vehicle routing problem. J Opl Res Soc 2000;51:564-73.
52. Schulze J, Fahle T. A parallel algorithm for the vehicle routing problem with time window constraints. Ann Oper Res 1999;86:585-607.

53. Tarantilis CD, Kiranoudis CT, Vassiliadis VS. A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. Eur J Opl Res 2004;152:148-58.
54. Toth P, Vigo D. An exact algorithm for the vehicle routing problem with backhauls. Transport Sci 1997;31:372-85.
55. Toth P, Vigo D. Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics 2002;123:487-512.
56. Van Breedam A. A parametric analysis of heuristics for the vehicle routing problem with side-constraints. Eur J Opl Res 2002;137:348-70.

**II. ELECTRONIC**

57. The VRP Web. [accessed November 2004] available at: http://neo.lcc.uma.es/radi-aeb/WebVRP/
58. Science Direct. [accessed November 2004 ] available at: http://sciencedirect.com/

**IV. OTHER**

59. Bekker J. Simulation of discrete-event stochastic processes. Unpublished course notes. Stellenbosch: University of Stellenbosch; 2003.
60. Bekker J. Some meta-heuristics for combinatorial optimization. Unpublished course notes. Stellenbosch: University of Stellenbosch; 2003.
61. Van Deventer PJU. Statistics for scientists 748. Unpublished course notes. Stellenbosch: University of Stellenbosch; 2003.
62. Van Vuuren JH. Introduction to graph theory. Unpublished course notes. Stellenbosch: University of Stellenbosch; 2003.

# Appendix A Distribution Definitions

This appendix contains the definitions of the various distributions as they were implemented in the software tool. The methods of random variate generation and parameter estimation are stated for each distribution where applicable.

## DEFINITION OF SYMBOLS

| | |
|---|---|
| Expected value (or mean) | $\mu$ |
| Variance | $\sigma 2$ |
| Standard deviation | $\sigma$ |
| Observed value | $x_i$ |
| Sample size | $n$ |
| Sample mean | $\bar{x} = n^{-1}\sum_{i=1i}^{n} x_i$ |
| Sample variance | $s^2 = (n-1)^{-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ |

## BETA AND GAMMA FUNCTIONS

Beta function: $B(v,w) = \int_0^1 u^{v-1}(1-u)^{w-1}du$

Gamma function: $\Gamma(c) = \int_0^\infty e^{-u}u^{c-1}du$

Relationships: $B(v,w) = B(w,v)$

$$B(v,w) = \frac{\Gamma(v)\Gamma(w)}{\Gamma(v+w)}$$

$\Gamma(c) = (c\text{-}1) \times \Gamma(c\text{-}1)!$

If v, w and c are integers: $B(v,w) = \frac{(v-1)!\times(w-1)!}{(v+w-1)!}$

$B(1,1)=1$

$B(½,½) = \pi$

$\Gamma(c) = \Gamma(c\text{-}1)!$

$\Gamma(0) = 1, \Gamma(1) = 1, \Gamma(½) = \pi\ ½$

**BETA DISTRIBUTION**

Variate: $\beta(v,w)$

Parameters: Shape parameter $v > 0$

Shape parameter $w > 0$

Range: $0 \leq x \leq 1$

Probability density function: $$f(x) = \frac{x^{v-1}(1-x)^{w-1}}{B(v,w)}$$

Mean: $\mu = v/(v+w)$

Variance: $\sigma^2 = uw/(v+w)^2(v+w+1)$

Parameter estimation: $$v \approx \bar{x}\left(\frac{\bar{x}(1-\bar{x})}{s^2} - 1\right)$$

$$w \approx (1-\bar{x})\left(\frac{\bar{x}(1-\bar{x})}{s^2} - 1\right)$$

Random variate generation: If v and w are integers, a Beta variate $\beta(v,w)$ can be generated using the unit rectangular variate **R** and two Gamma variates $\gamma(1,v)$ and $\gamma(1,w)$ so that:

$$\beta(v,w) \approx \frac{\gamma(1,v)}{\gamma(1,v)+\gamma(1,w)}$$

**CHI-SQUARED DISTRIBUTION**

Variate: $\chi^2(v)$

Parameters: Shape parameter $v > 0$ representing the degrees of freedom

Range: $0 \leq x \leq \infty$

Probability density function: $$f(x) = \frac{x^{(v-2)/2}e^{-x/2}}{2^{v/2}\Gamma(v/2)}$$

Mean: $\mu = v$

Variance: $\sigma^2 = 2v$

Standard deviation: $\sigma = (2v)^{1/2}$

**ERLANG DISTRIBUTION**

The Erlang variate is a Gamma variate with an integer shape parameter. If the shape parameter is 1, then the Erlang variate is an Exponential variate. See "Gamma Distribution" for further details.

**EXPONENTIAL DISTRIBUTION**

Variate: **Exp**($b$)

Parameters: Scale parameter $b > 0$ representing the mean

Alternatively the hazard rate, $\lambda$. ($\lambda = b^{-1}$)

Range: $0 \leq x \leq \infty$

Probability density function: $f(x) = b^{-1}e^{-x/b}$

Distribution Function: $F(x) = 1 - e^{-x/b}$

Mean: $\mu = b$

Variance: $\sigma^2 = b^2$

Standard deviation: $\sigma = b$

Parameter estimation: $b \approx \bar{x}$

Random variate generation: An Exponential variate can be generated using the unit rectangular variate **R** so that:

**Exp**($b$) ≈ -$b$.Log(**R**)

**GAMMA DISTRIBUTION**

Variate: $\Gamma(b,c)$

Parameters: Scale parameter $b > 0$. Alternatively $\lambda$. ($\lambda = b^{-1}$)

Shape parameter $c > 0$.

Range: $0 \leq x \leq \infty$

Probability density function: $f(x) = \left(\frac{x}{b}\right)^{c-1} \cdot \frac{e^{-x/b}}{b.\Gamma(c)}$

Distribution function: $F(x) = 1 - \left(e^{-x/b}\right)\left(\sum_{i=0}^{c-1} \frac{(x/b)^i}{i!}\right)$

Mean: $\mu = bc$

Variance: $\sigma^2 = b^2c$

Standard deviation: $\sigma = bc^{½}$

Parameter estimation: $b \approx s^2/\bar{x}$

$c \approx \left(\bar{x}/s\right)^2$

Random variate generation: If $c$ is an integer, a Gamma variate (or Erlang variate) can be generated using a series of unit rectangular variates $\mathbf{R}_i$ ($i$ = 1, 2, ... , $c$) so that:

$$\Gamma(b,c) \approx -b.Log\left(\prod_{i=1}^{c} R_i\right)$$

**LOGNORMAL DISTRIBUTION**

Variate: **L**$(m,c)$

Parameters: Scale parameter $m > 0$ representing the median. Alternatively $b$, the mean of Log(**L**). $m = e^b$.

Shape parameter c > 0 representing the standard deviation of Log(**L**).

Range: $0 \leq x \leq \infty$

Probability density function: $$f(x) = \frac{1}{xc\sqrt{2\pi}} \exp\left(\frac{-(\log(x/m))^2}{2c^2}\right)$$

Mean: $\mu = m.\exp(½.c^2)$

Variance: $\sigma^2 = m^2.\exp(c^2).[1- \exp(c^2)]$

Parameter estimation: $m \approx e^b$

$$b = (n^{-1})\sum_{i=1}^{n} \log(x_i)$$

$$c^2 \approx ((n-1)^{-1})\sum_{i=1}^{n} (\log(x_i) - b)^2$$

Random variate generation: An Lognormal variate can be generated using the unit normal variate **N** so that:

$\mathbf{L}(m,c) \approx m.e^{cN}$

**NORMAL DISTRIBUTION**

Variate: **N**$(b,c)$. If $b$ = 0 and $c$ = 1, the variate is the unit normal distribution **N**.

Parameters: Location parameter $b$, representing the mean.

Scale parameter $c > 0$ representing the standard deviation. Alternatively $c^2$, the variance, can be used as a scale parameter.

Range: $-\infty \leq x \leq \infty$

Probability density function: $$f(x) = \frac{1}{\sqrt{2c^2\pi}} \exp\left[\frac{-(x-b)^2}{2c^2}\right]$$

Mean: $\mu = b$

Variance: $\sigma^2 = c^2$

Standard deviation: $\sigma = c$

Parameter estimation: $b \approx \bar{x}$

$c^2 \approx ns^2/(n\text{-}1)$

If $n$ is large then $c^2 \approx s^2$

Random variate generation: A unit normal variate was be generated using a series of unit rectangular variates **R***i* ($i$ = 1, 2, ..., $k$) so that:

$$N \approx \frac{\sum_{i=1}^{k} R_i - k/2}{\sqrt{k/12}}$$

For many purposes it is sufficient to use $k$ = 12 (so that the equation above simplifies), but in this project $k$ = 36 was used.

Although better methods exist for the generation of a unit normal variate (e.g. the polar method), it was deemed adequate to use the simple method above.

**RECTANGULAR DISTRIBUTION**

Variate: **R**($a$,$b$). If $a$ = 0 and $b$ = 1 we refer to the variate as the unit rectangular variate, **R**.

Parameters: Location parameter $a$, representing the lower limit of the range. Scale parameter $b > 0$. Alternatively $h$ ($h > a$), representing the upper limit of the range. $h = a + b$

Range: $a \le x \le h = a + b$

Probability density function: $f(x) = 1/b$

Distribution function: $F(x) = (x - a)/b$

Mean: $\mu = a + b/2$

Variance: $\sigma^2 = b^2/12$

Standard deviation: $\sigma = b/12^{1/2}$

Parameter estimation: $a \approx \bar{x} - \sqrt{3}s$

$b \approx \sqrt{12}s$

Random variate generation: There are various random variate generators. The linear congruential generators produce a sequence of integers $Z_i$ using the formula: $Z_i = (x.Z_{i-1} + y) \bmod m$.

Appropriate values for the modulus m, the multiplier $x$, the increment $y$ and the seed $Z_0$ must be chosen, in order to prevent the generator from degenerating.

These numbers are then converted to a unit rectangular variate by: $R_i = Z_i/m$.

Although this is an effective method to generate a unit rectangular variate, Visual Basic 6's generator was used in this project.

**WEIBULL DISTRIBUTION**

Variate: $\mathbf{W}(b,c)$

Parameters: Scale parameter $b > 0$ (sometimes referred to as the characteristic life)

Shape parameter $c > 0$

Range: $0 \leq x \leq \infty$

Probability density function: $$f(x) = \frac{cx^{c-1}}{b^c} \exp\left(-\left(\frac{x}{b}\right)^c\right)$$

Distribution function: $$F(x) = 1 - \exp\left(-\left(\frac{x}{b}\right)^c\right)$$

Mean: $$\mu = b\Gamma\left(\frac{c+1}{c}\right)$$

Variance: $$\sigma^2 = b^2\left[\Gamma\left(\frac{c+2}{c}\right) - \left(\Gamma\left(\frac{c+1}{c}\right)\right)^2\right]$$

Parameter estimation: The two parameters defining the variate has to be estimated iteratively using the following equations:

$$b \approx \left[n^{-1}\sum_{i=1}^{n} x_i^c\right]^{\frac{1}{c}} \text{ and } c \approx \frac{n}{(1/b)^c \sum_{i=1}^{n} x_i^c \log x_i - \sum_{i=1}^{n} \log x_i}$$

Random variate generation: An Weibull variate can be generated using the unit rectangular variate **R** so that:

$\mathbf{W}(b,c) \approx b(-\log \mathbf{R})^{1/c}$

# Appendix B Chi-Squared Test

The chi-squared goodness-of-fit test is used by the data analysis structure when examining data describing a continuous random variable. The following procedure is followed by the data analysis structure when a test is performed:

1. The user specifies the theoretical distribution he/she wants to test. The user can use the values estimated by the data analysis structure for the parameters of the tested theoretical distribution or use his/her chosen values.

2. The user is prompted for a level of significance for the test (represented by α).

3. The number of observations in the text file is counted and the suggested number of intervals for the data classes is calculated by Sturges's rule:

   $$k = \lfloor 1 + \log_2 n \rfloor$$

   where $k$ is the number of categories and $n$ the number of observations.

   The ranges of the intervals are determined by evenly spacing the intervals over the observed data range, although it would be more desirable to space the intervals in a way that equalises the area under the tested theoretical distribution curve for all intervals. (The implementations of such ranges are cumbersome in the Visual Basic environment; therefore evenly-spaced ranges are used.)

   Two more intervals are then added on either side to cover the entire definition range of the tested theoretical distribution.

4. The number of actual observations ($O_i$) in each interval is counted and the number of expected observations ($E_i$) in each interval is determined. The number of expected observations is calculated as:

   $$E_i = n.p_i$$

   Where $n$ is the number of observations and $p_i$ is the proportion of the $i^{th}$ interval as set by the probability density function of the tested theoretical distribution.

The proportion of each interval was calculated using the trapezoidal rule that states that $\int_a^b f(x)dx$ can be approximated by:

$$A = \frac{h}{2}\left(y_0 + 2y_1 + 2y_2 + \ldots + 2y_{j-1} + y_j\right)$$

where $j$ is the number of sub-intervals and $y_\beta = f(x_\beta)$. A thousand sub-intervals ($j$=1000) were used for each proportion estimated.

Simpson's one-third rule can also be used to for numeric integration, but it was found that the trapezoidal rule where a sufficiently accurate approximation.

5. Intervals are then repeatedly grouped together while there are more than 3 intervals and there is an interval with an expected number of observations less than 5 ($E_i$<5). The interval with $E_i$<5 is joined with the interval on its right hand side (unless of course it is the interval on the far right, in which case it is joined to the interval on its left).

6. The calculated chi-squared value is then determined:

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

The chi-squared critical value, $\chi^2_{critical}$, is determined by repeated numeric integration of the chi-squared distribution over a wider area until the area under the curve is larger than 1 – α. Again the trapezoidal rule is used for the numeric integration. It is useful to note that if α is larger than 0.5 the distribution can be integrated from $v$, the number of degrees of freedom, since the mean of the distribution is $v$.

In order to numerically integrate the chi-squared distribution, the gamma function must also be evaluated. If the degrees of freedom ($v$) are even, numeric integration of the gamma function can be avoided. However, if $v$ is uneven and small, numeric integration of the gamma function is required and it will result in a rounding error. When $v$ is greater than 1 the rounding error becomes insignificant.

The degrees of freedom can be calculated as follows:

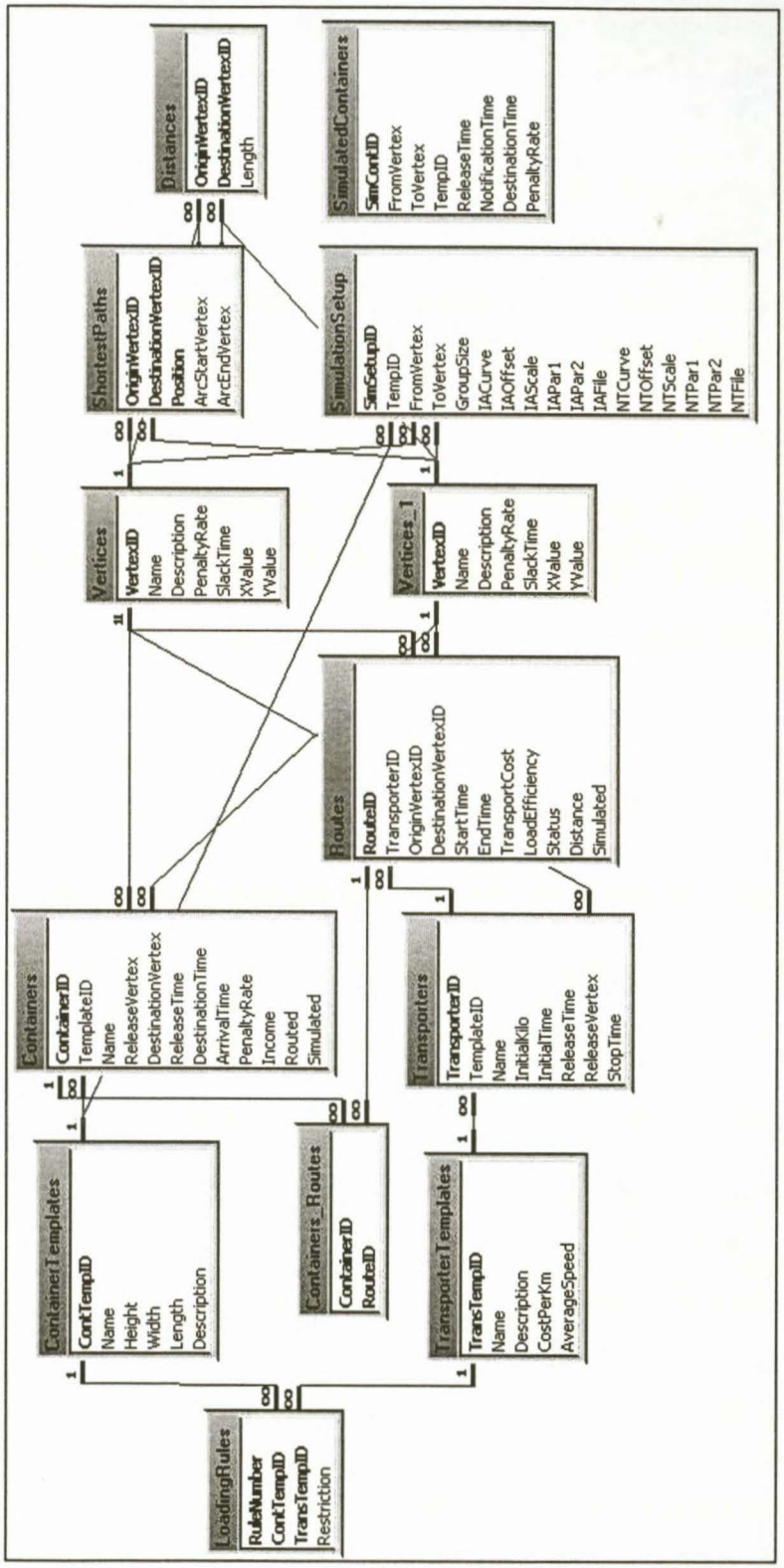
$$v = k - m - 1$$

where $k$ is the number of intervals and $m$ is the number of estimated parameters of the tested theoretical distribution.

7. If $\chi^2_{calculated} > \chi^2_{critical}$ there is sufficient evidence to reject the hypothesis that the observed data is from a theoretical distribution as specified by the user, with a level of significance α. If $\chi^2_{calculated} \leq \chi^2_{critical}$ there is not sufficient evidence to reject the hypothesis.

# Appendix C Database Design

The diagram below shows the complete EERD of the database:



The following table relates the entities of the generic problem with the tables of the database.

| **Relationship between entities of the generic problem and the tables in the database** | |
|---|---|
| Entities of the Generic Problem | Tables in the Database |
| Transporter Template | TransporterTemplates |
| Transporter (Instance) | Transporters |
| Container Templates | ContainerTemplates |
| Container (Instance) | Containers |
| Vertex | Vertices |
| Arc | Distances |
| Movements | Routes |
| Loading Rules | LoadingRules |
| | |
| **Other tables required** | |
| Table in the Database | Explanation |
| Containers_Routes | The association of a container with various movements are stored in this table. |
| ShortestPaths | The shortest paths between all vertices are stored in this table so that the paths do not have to be calculated repeatedly. |
| SimulationSetup | This table stores the configuration of container entering patterns for the simulation model. |
| SimulatedContainers | This table is used as a temporary storage for the containers created during a simulation run. |

The following table contains a data dictionary of the database.

| **Data Dictionary** | | |
|---|---|---|
| **Table/Field** | **Type** | **Purpose** |
| | | |
| Containers | Table | Stores container entities |
| ContainerID | Long Integer | Primary key |
| TemplateID | Long Integer | Links container template entity |
| Name | String(255) | Stores user identifier (e.g. a serial number) |
| ReleaseVertex | Long Integer | Links release vertex entity |
| DestinationVertex | Long Integer | Links destination vertex entity |
| ReleaseTime | Date/Time | Stores release time |
| DestinationTime | Date/Time | Stores destination time |
| PenaltyRate | Currency | Stores penalty rate [R/h] |
| Income | Currency | Stores income received for moving container [R] |
| Routed | Boolean | Container routed to its destination. (Note: Data duplicated to speed up simulations.) |
| Simulated | Boolean | Simulated or actual container |
| ArrivalTime | Date/Time | Stores container arrival time (Note: Data duplicated to speed up simulations.) |
| | | |

| ContainerTemplates | Table | Stores container template entities |
|---|---|---|
| ContTempID | Long Integer | Primary Key |
| Name | String(255) | Stores user identifier (e.g. a serial number) |
| Height | Integer | Stores container template attribute |
| Width | Integer | Stores container template attribute |
| Length | Integer | Stores container template attribute |
| Description | String(255) | Stores a description of the template (Optional) |
| | | |
| Containers_Routes | Table | Intersection table to assign containers entities to movement entities |
| ContainerID | Long Integer | Foreign key |
| RouteID | Long Integer | Foreign key |
| | | |
| Distances | Table | Stores the arc entities of the transportation network |
| OriginVertexID | Long Integer | Links vertex entity |
| DestinationVertexID | Long Integer | Links vertex entity |
| Length | Long Integer | Arc distance [km] |
| | | |
| LoadingRules | Table | Stores the loading rules of transporter templates |
| RuleNumber | Long Integer | Identifies each rule of a transporter |
| ContTempID | Long Integer | Links container template entity |
| TransTempID | Long Integer | Links transporter template entity |
| Restriction | Long Integer | Loading restriction |
| | | |
| Routes | Table | Stores the movement entities |
| RouteID | Long Integer | Primary key |
| TransporterID | Long Integer | Links transporter entity |
| OriginVertexID | Long Integer | Links origin vertex entity |
| DestinationVertexID | Long Integer | Links destination vertex entity |
| StartTime | Date/Time | Stores start time of movement |
| EndTime | Date/Time | Stores end time of movement |
| TransportCost | Currency | Stores transit cost of transporter travelling on movement |
| LoadEfficiency | Single | Stores the transporter efficiency of transporter travelling on movement |
| Status | Byte | Stores status of movement. 1=Assigned, 2=Fixed, 3=Locked, 4=Completed |
| Distance | Long Integer | Distance of arc associated with movement |
| Simulated | Boolean | Actual or simulated movement |
| | | |
| ShortestPaths | Table | Stores the shortest paths between all vertex pairs so that is does not have to be calculated repeatedly |
| OriginVertexID | Long Integer | Stores origin vertex of path |
| DestinationVertexID | Long Integer | Stores destination vertex of path |
| Position | Long Integer | Stores where the edge fits into the path |
| ArcStartVertex | Long Integer | Stores the edge's origin vertex |
| ArcEndVertex | Long Integer | Stores the edge's destination vertex |
| | | |
| SimulatedContainers | Table | Used as a temporary storage for the containers created during a simulation run. |
| SimContID | Long Integer | Primary key |
| FromVertex | Long Integer | Stores simulated container origin vertex |
| ToVertex | Long Integer | Stores simulated container destination vertex |
| TempID | Long Integer | Stores simulated container template |
| ReleaseTime | Date/Time | Stores time when container is released |
| NotificationTine | Date/Time | Stores time when container must enter the system |
| DestinationTime | Date/Time | Stores simulated container destination time |
| PenaltyRate | Single | Stores simulated container penalty rate |

| | | |
|---|---|---|
| SimulationSetup | Table | Stores the configuration of container entering patterns for the simulation model |
| SimSetupID | Long Integer | Primary key |
| TempID | Long Integer | Stores container template of entering pattern |
| FromVertex | Long Integer | Stores group release vertex |
| ToVertex | Long Integer | Stores group destination vertex |
| GroupSize | Long Integer | Stores group size |
| IACurve | String(15) | Stores inter-arrival time distribution |
| IAOffset | Single | Stores offset of distribution |
| IAScale | Single | Stores scale factor of distribution |
| IAPar1 | Single | Stores first parameter of distribution |
| IAPar2 | Single | Stores second parameter of distribution |
| IAFile | String(255) | Stores text file name for inter-arrival times |
| NTCurve | String(15) | Stores notification time distribution |
| NTOffset | Single | Stores offset of distribution |
| NTScale | Single | Stores scale factor of distribution |
| NTPar1 | Single | Stores first parameter of distribution |
| NTPar2 | Single | Stores second parameter of distribution |
| NTFile | String(255) | Stores text file name for notification times |
| | | |
| Transporters | Table | Stores the transporter entities |
| TransporterID | Long Integer | Primary key |
| TemplateID | Long Integer | Links transporter template entity |
| Name | String(255) | Stores user identifier (e.g. a truck number) |
| InitialKilo | Long Integer | Stores initial kilometre reading of transporter [km] |
| InitialTime | Date/Time | Stores time when kilometre reading was taken |
| ReleaseTime | Date/Time | Stores release time of the transporter |
| ReleaseVertex | Date/Time | Links release vertex |
| StopTime | Date/Time | Stores halt time of the transporter |
| | | |
| Transporter Templates | Table | Stores the transporter template entities |
| TransTempID | Long Integer | Primary key |
| Name | String(255) | Stores user identifier (e.g. a truck type) |
| Description | String(255) | Stores a description of the template (Optional) |
| CostPerKm | Single | Stores the running cost of the transporter template [R/km] |
| AverageSpeed | Single | Stores the average speed of the transporter [km/h] |
| | | |
| Vertices | Table | Stores the vertex entities |
| VertexID | Long Integer | Primary key |
| Name | String(255) | Stores user identifier (e.g. a city acronym) |
| Description | String(255) | Stores a description of the vertices (Optional) |
| PenatlyRate | Currency | Stores the default penalty rate of containers released from the vertex [R/h] |
| SlackTime | Integer | Stores the default slack time of containers released from the vertex [R/h] |
| XValue | Long Integer | Stores x position of vertex on 1000x1000 map |
| YValue | Long Integer | Stores y position of vertex on 1000x1000 map |

# Appendix D Simulation Output

This appendix contains a summary of the output of the various simulations executed.

## D.1. STUDY IN COMPARING HEURISTICS

### D.1.1 Scheduling for a maximum service level

Fifteen replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result:

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 0 | 3,107,729 | 3,107,729 | 100.00 | n/a | 36.56 |
| 2 | 0 | 3,091,442 | 3,091,442 | 100.00 | n/a | 34.57 |
| 3 | 0 | 3,083,912 | 3,083,912 | 100.00 | n/a | 34.47 |
| 4 | 0 | 3,121,690 | 3,121,690 | 100.00 | n/a | 34.49 |
| 5 | 0 | 3,040,570 | 3,040,570 | 100.00 | n/a | 36.59 |
| 6 | 0 | 3,047,380 | 3,047,380 | 100.00 | n/a | 36.27 |
| 7 | 0 | 3,059,169 | 3,059,169 | 100.00 | n/a | 36.41 |
| 8 | 0 | 3,153,904 | 3,153,904 | 100.00 | n/a | 34.48 |
| 9 | 0 | 3,083,961 | 3,083,961 | 100.00 | n/a | 35.89 |
| 10 | 0 | 3,098,901 | 3,098,901 | 100.00 | n/a | 34.46 |
| 11 | 0 | 3,035,781 | 3,035,781 | 100.00 | n/a | 34.28 |
| 12 | 0 | 3,112,213 | 3,112,213 | 100.00 | n/a | 35.89 |
| 13 | 0 | 3,185,265 | 3,185,265 | 100.00 | n/a | 34.27 |
| 14 | 0 | 3,111,994 | 3,111,994 | 100.00 | n/a | 36.35 |
| 15 | 0 | 3,083,572 | 3,083,572 | 100.00 | n/a | 35.44 |
| **Averages** | | **3,094,499** | **3,094,499** | **100.00** | **n/a** | **35.36** |
| **95% Conf. half width** | | **<1%** | **<1%** | **n/a** | **n/a** | **< 2 %** |

### D.1.2 Scheduling for a minimum total cost ($\delta$ = 10)

Ten replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with $\delta$ = 10):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 810,508 | 1,447,401 | 2,257,910 | 46.07 | 73.07 | 80.24 |
| 2 | 765,351 | 1,532,565 | 2,297,916 | 49.59 | 73.12 | 75.71 |
| 3 | 1,013,726 | 1,444,315 | 2,458,041 | 41.19 | 78.39 | 77.85 |
| 4 | 903,529 | 1,466,474 | 2,370,003 | 45.30 | 80.75 | 77.76 |
| 5 | 1,203,641 | 1,552,921 | 2,756,562 | 44.59 | 104.91 | 77.85 |
| 6 | 939,378 | 1,516,745 | 2,456,124 | 46.46 | 84.87 | 77.26 |
| 7 | 888,123 | 1,408,040 | 2,296,164 | 43.72 | 73.69 | 78.31 |
| 8 | 968,809 | 1,326,664 | 2,295,474 | 45.39 | 83.29 | 80.87 |
| 9 | 841,867 | 1,524,997 | 2,366,865 | 47.31 | 76.37 | 76.45 |
| 10 | 1,041,733 | 1,478,014 | 2,519,747 | 38.84 | 81.61 | 76.18 |
| **AVE.** | **937,667** | **1,469,814** | **2,407,481** | **44.85** | **81.00** | **77.85** |
| **Range** | **< 10 %** | **< 4 %** | **< 5 %** | **< 5 %** | **< 10 %** | **< 2 %** |

### D.1.3 Scheduling for a minimum total cost (δ = 15)

Ten replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with δ = 15):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 1,025,574 | 1,503,554 | 2,529,128 | 38.51 | 75.63 | 75.51 |
| 2 | 1,073,617 | 1,483,080 | 2,556,698 | 37.10 | 77.72 | 76.48 |
| 3 | 1,010,410 | 1,393,054 | 2,403,465 | 50.54 | 101.8 | 81.20 |
| 4 | 648,879 | 1,551,354 | 2,200,234 | 50.08 | 65.73 | 77.90 |
| 5 | 803,117 | 1,489,410 | 2,292,528 | 44.48 | 67.79 | 76.20 |
| 6 | 893,493 | 1,484,143 | 2,377,637 | 48.87 | 78.66 | 80.36 |
| 7 | 949,160 | 1,505,818 | 2,454,978 | 43.10 | 80.76 | 81.12 |
| 8 | 760,295 | 1,419,524 | 2,179,820 | 54.45 | 82.15 | 79.34 |
| 9 | 751,323 | 1,459,908 | 2,211,232 | 52.17 | 79.37 | 76.44 |
| 10 | 931,668 | 1,505,200 | 2,436,868 | 42.37 | 75.27 | 79.85 |
| **AVE.** | **884,754** | **1,479,504** | **2,364,258** | **41.17** | **78.49** | **78.44** |
| **Range** | **< 12 %** | **< 3 %** | **< 5 %** | **< 10 %** | **< 10 %** | **< 2 %** |

### D.1.4 Scheduling for a minimum total cost (δ = 25)

Ten replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with δ = 25):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 974,532 | 1,398,869 | 2,373,401 | 40.70 | 76.33 | 77.76 |
| 2 | 851,311 | 1,520,558 | 2,371,870 | 45.46 | 75.73 | 75.34 |
| 3 | 967,882 | 1,472,695 | 2,440,577 | 44.02 | 82.93 | 77.98 |
| 4 | 995,618 | 1,420,201 | 2,415,820 | 41.38 | 80.49 | 75.01 |
| 5 | 948,056 | 1,534,546 | 2,482,602 | 38.34 | 72.38 | 77.59 |
| 6 | 1,050,549 | 1,351,925 | 2,402,474 | 44.34 | 88.97 | 80.41 |
| 7 | 1,125,573 | 1,491,601 | 2,617,174 | 40.28 | 91.62 | 76.81 |
| 8 | 812,922 | 1,509,450 | 2,322,373 | 49.51 | 72.33 | 77.19 |
| 9 | 1,221,232 | 1,447,873 | 2,669,105 | 38.66 | 94.66 | 77.16 |
| 10 | 776,366 | 1,469,996 | 2,246,363 | 44.40 | 67.50 | 77.70 |
| **AVE.** | **972,404** | **1,461,771** | **2,434,175** | **42.71** | **80.29** | **77.30** |
| **Range** | **< 12 %** | **< 3 %** | **< 3 %** | **< 6 %** | **< 10 %** | **< 2 %** |

### D.1.5 Scheduling for a minimum total cost (δ = 150)

Ten replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with δ = 150):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 525,466 | 1,669,069 | 2,194,535 | 49.05 | 50.72 | 76.57 |
| 2 | 483,264 | 1,763,276 | 2,246,540 | 56.35 | 53.83 | 74.51 |
| 3 | 512,166 | 1,591,710 | 2,103,876 | 53.57 | 52.83 | 73.29 |
| 4 | 544,176 | 1,749,886 | 2,294,062 | 52.10 | 55.68 | 72.58 |
| 5 | 460,166 | 1,670,572 | 2,130,738 | 53.56 | 47.81 | 75.68 |
| 6 | 505,940 | 1,568,271 | 2,074,211 | 50.05 | 48.58 | 71.28 |
| 7 | 499,387 | 1,644,545 | 2,143,932 | 53.27 | 49.40 | 75.22 |
| 8 | 499,355 | 1,741,544 | 2,240,899 | 54.49 | 52.52 | 71.51 |
| 9 | 585,029 | 1,636,036 | 2,221,065 | 48.55 | 56.89 | 70.79 |
| 10 | 520,253 | 1,801,862 | 2,322,115 | 52.14 | 52.10 | 72.70 |
| **AVE.** | 513,520 | 1,683,677 | 2,197,197 | 52.31 | 52.04 | 73.41 |
| **Range** | < 5 % | < 5 % | < 3 % | < 5 % | < 5 % | < 2 % |

### D.1.6 Scheduling for a minimum total cost (δ = 300)

Ten replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with δ = 300):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 259,788 | 1,911,121 | 2,170,909 | 61.88 | 32.43 | 65.16 |
| 2 | 248,619 | 1,721,270 | 1,969,889 | 61.26 | 31.27 | 68.19 |
| 3 | 258,099 | 1,694,239 | 1,952,338 | 61.76 | 33.08 | 71.95 |
| 4 | 265,347 | 2,003,770 | 2,269,117 | 60.54 | 31.02 | 62.85 |
| 5 | 277,678 | 1,873,519 | 2,151,197 | 60.56 | 33.85 | 67.51 |
| 6 | 261,180 | 1,924,845 | 2,186,025 | 59.88 | 31.25 | 65.35 |
| 7 | 257,639 | 1,851,493 | 2,109,132 | 61.81 | 31.69 | 68.31 |
| 8 | 275,623 | 1,726,594 | 2,002,217 | 59.64 | 33.66 | 70.68 |
| 9 | 299,864 | 1,906,257 | 2,206,121 | 57.66 | 32.75 | 66.35 |
| 10 | 252,795 | 1,698,100 | 1,950,895 | 59.91 | 31.68 | 70.80 |
| **AVG.** | 265,663 | 1,831,121 | 2,096,784 | 60.49 | 32.27 | 67.72 |
| **Range** | < 5 % | < 5 % | < 5 % | < 2 % | < 3 % | < 3 % |

### D.1.7 Scheduling for a minimum total cost (δ = 600)

Ten replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with δ = 600):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 180,923 | 1,907,744 | 2,088,667 | 71.62 | 28.38 | 64.79 |
| 2 | 197,436 | 1,818,419 | 2,015,855 | 70.96 | 29.04 | 68.33 |
| 3 | 160,042 | 1,979,846 | 2,139,888 | 71.01 | 28.99 | 64.24 |
| 4 | 163,562 | 1,824,898 | 1,988,460 | 69.96 | 30.04 | 66.67 |
| 5 | 187,762 | 1,984,794 | 2,172,556 | 69.40 | 30.60 | 62.11 |
| 6 | 208,674 | 1,885,815 | 2,094,489 | 67.66 | 32.34 | 64.58 |
| 7 | 153,576 | 1,952,107 | 2,105,683 | 69.83 | 30.17 | 61.26 |
| 8 | 178,315 | 1,877,529 | 2,055,844 | 74.84 | 25.16 | 67.10 |
| 9 | 155,900 | 1,934,457 | 2,090,357 | 69.42 | 30.58 | 62.96 |
| 10 | 204,518 | 1,842,726 | 2,047,244 | 65.63 | 34.37 | 66.64 |
| **AVE.** | 179,071 | 1,900,834 | 2,079,904 | 70.03 | 29.97 | 64.87 |
| **Range** | < 8 % | < 3 % | < 2 % | < 3 % | < 6 % | < 3 % |

### D.1.8 Scheduling for a minimum total cost (δ = 2000)

Ten replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with δ = 2000):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 107,102 | 1,821,468 | 1,928,570 | 86.85 | 13.15 | 67.71 |
| 2 | 99,316 | 1,897,418 | 1,996,734 | 84.79 | 15.21 | 63.78 |
| 3 | 121,977 | 1,868,107 | 1,990,084 | 83.14 | 16.86 | 62.51 |
| 4 | 120,755 | 1,902,921 | 2,023,676 | 83.73 | 16.27 | 63.59 |
| 5 | 150,567 | 1,949,469 | 2,100,036 | 82.81 | 17.19 | 65.37 |
| 6 | 102,271 | 1,848,688 | 1,950,959 | 84.81 | 15.19 | 64.33 |
| 7 | 125,315 | 1,841,163 | 1,966,479 | 86.14 | 13.86 | 63.07 |
| 8 | 106,867 | 1,855,790 | 1,962,657 | 83.96 | 16.04 | 65.65 |
| 9 | 139,508 | 1,765,206 | 1,904,714 | 83.01 | 16.99 | 66.69 |
| 10 | 110,495 | 1,965,567 | 2,076,062 | 85.85 | 14.15 | 61.24 |
| **AVE.** | 118,417 | 1,871,580 | 1,989,997 | 84.51 | 15.49 | 64.39 |
| **Range** | < 11 % | < 3 % | < 3 % | < 2 % | < 8 % | < 3 % |

## D.2. STUDY IN CHANGING THE DELAY TIME

Ten replications were executed beyond the warm-up period and the weekly parameters determined for a delay time of 1.5 hours. The table below summarises the result for the minimum total cost heuristic (with $\delta = 10$):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 656,807 | 1,601,317 | 2,258,125 | 53.85 | 67.14 | 75.99 |
| 2 | 932,089 | 1,614,221 | 2,546,311 | 47.49 | 83.17 | 77.05 |
| 3 | 804,380 | 1,500,101 | 2,304,482 | 54.43 | 82.01 | 78.97 |
| 4 | 812,255 | 1,499,493 | 2,311,749 | 50.65 | 77.80 | 76.60 |
| 5 | 791,526 | 1,594,692 | 2,386,218 | 46.24 | 68.49 | 77.28 |
| 6 | 580,240 | 1,676,765 | 2,257,006 | 51.56 | 52.96 | 76.58 |
| 7 | 894,361 | 1,608,151 | 2,502,513 | 49.49 | 81.54 | 77.19 |
| 8 | 689,944 | 1,489,380 | 2,179,324 | 51.21 | 68.42 | 78.93 |
| 9 | 782,343 | 1,511,278 | 2,293,622 | 56.30 | 88.01 | 81.61 |
| 10 | 837,814 | 1,599,780 | 2,437,595 | 50.32 | 79.12 | 78.40 |
| **AVE.** | **778,176** | **1,569,519** | **2,347,695** | **51.15** | **74.87** | **77.86** |
| **Range** | **< 10 %** | **< 3 %** | **< 4 %** | **< 5 %** | **< 10 %** | **< 2 %** |

## D.3. STUDY IN CHANGING THE CARRIER FLEET

### D.3.1 Decreasing the fleet size with 50%

The fleet size was decreased with 50% and 14 runs executed with the maximum service level algorithm. The table below summarises the result of the weekly parameters.

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 0 | 1,772,517 | 1,772,517 | 100 | n/a | 61.45 |
| 2 | 0 | 1,801,424 | 1,801,424 | 100 | n/a | 61.86 |
| 3 | 0 | 1,788,952 | 1,788,952 | 100 | n/a | 60.50 |
| 4 | 0 | 1,789,819 | 1,789,819 | 100 | n/a | 57.56 |
| 5 | 0 | 1,780,352 | 1,780,352 | 100 | n/a | 59.90 |
| 6 | 0 | 1,800,159 | 1,800,159 | 100 | n/a | 59.83 |
| 7 | 0 | 1,768,528 | 1,768,528 | 100 | n/a | 61.86 |
| 8 | 0 | 1,817,395 | 1,817,395 | 100 | n/a | 61.88 |
| 9 | 0 | 1,792,102 | 1,792,102 | 100 | n/a | 61.21 |
| 10 | 0 | 1,791,102 | 1,791,102 | 100 | n/a | 62.59 |
| 11 | 0 | 1,803,865 | 1,803,865 | 100 | n/a | 60.47 |
| 12 | 0 | 1,778,796 | 1,778,796 | 100 | n/a | 61.90 |
| 13 | 0 | 1,792,395 | 1,792,395 | 100 | n/a | 61.51 |
| 14 | 0 | 1,776,914 | 1,776,914 | 100 | n/a | 61.45 |
| **Averages** | | **1,789,594** | **1,789,594** | **100** | **n/a** | **61.00** |
| **95% Conf. half width** | | **< 1 %** | **< 1 %** | **n/a** | **n/a** | **< 2 %** |

One run was executed with the MTC algorithm (with δ=10) for larger fleet. The simulation run time became too long to create 10 replications beyond the warm-up period. The table below summarises the result of the daily parameters of one replication.

| Run | Penalty Cost | Running Cost | Total Cost | Service Level [%] | Ave. Late Time [h] | Transporter Efficiency |
|---|---|---|---|---|---|---|
| 04/01/01 | 0 | 81,808 | 81,808 | 100 | 0 | 0 |
| 04/01/02 | 0 | 130,692 | 130,692 | 100 | 0 | 13.76 |
| 04/01/03 | 33,772 | 95,729 | 129,501 | 74.86 | 65.06 | 38.70 |
| 04/01/04 | 29,369 | 135,472 | 164,841 | 75.12 | 44.42 | 65.64 |
| 04/01/05 | 96,172 | 132,966 | 229,138 | 58.82 | 78.99 | 78.40 |
| 04/01/06 | 95,816 | 128,295 | 224,111 | 50.74 | 58.48 | 84.78 |
| 04/01/07 | 173,492 | 120,005 | 293,497 | 36.29 | 84.32 | 88.76 |
| 04/01/08 | 234,574 | 144,043 | 378,617 | 29.89 | 102.38 | 81.54 |
| 04/01/09 | 231,838 | 105,055 | 336,893 | 31.44 | 130.15 | 67.19 |
| 04/01/10 | 334,779 | 122,017 | 456,796 | 31.36 | 146.92 | 78.76 |
| 04/01/11 | 329,722 | 148,136 | 477,858 | 33.10 | 158.33 | 84.47 |
| 04/01/12 | 348,702 | 158,667 | 507,369 | 30.77 | 177.24 | 84.90 |
| 04/01/13 | 278,360 | 157,907 | 436,267 | 39.18 | 172.79 | 79.66 |
| 04/01/14 | 320,439 | 168,214 | 488,653 | 35.23 | 157.07 | 66.73 |
| 04/01/15 | 305,794 | 180,823 | 486,617 | 33.68 | 142.97 | 90.77 |
| 04/01/16 | 380,024 | 189,306 | 569,330 | 32.49 | 183.23 | 88.73 |
| 04/01/17 | 267,946 | 173,730 | 441,676 | 36.50 | 149.49 | 59.57 |
| 04/01/18 | 260,771 | 189,041 | 449,812 | 32.55 | 115.07 | 87.34 |
| 04/01/19 | 328,976 | 215,355 | 544,331 | 27.06 | 168.97 | 85.14 |
| 04/01/20 | 389,110 | 169,390 | 558,500 | 30.30 | 176.23 | 78.98 |
| 04/01/21 | 360,714 | 198,945 | 559,659 | 30.83 | 188.00 | 93.26 |
| 04/01/22 | 215,598 | 242,838 | 458,436 | 37.93 | 194.55 | 79.11 |

The fleet size was decreased with 50% and 10 runs executed with the MTC(δ=2000) algorithm. The table below summarises the result of the weekly parameters.

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | *686,507* | *1,436,147* | *2,122,654* | *38.78* | *61.22* | *79.09* |
| 2 | 490,626 | 1,517,061 | 2,007,687 | 51.05 | 48.95 | 79.19 |
| 3 | 454,200 | 1,538,637 | 1,992,837 | 48.39 | 51.61 | 76.72 |
| 4 | 390,079 | 1,588,086 | 1,978,165 | 51.80 | 48.20 | 77.35 |
| 5 | 312,813 | 1,500,753 | 1,813,566 | 56.20 | 43.80 | 77.94 |
| 6 | 231,364 | 1,500,628 | 1,731,992 | 61.80 | 38.20 | 75.87 |
| 7 | 527,669 | 1,534,765 | 2,062,434 | 48.78 | 51.22 | 78.41 |
| 8 | 451,278 | 1,552,435 | 2,003,713 | 50.03 | 49.97 | 77.45 |
| 9 | 495,686 | 1,452,590 | 1,948,276 | 50.83 | 49.17 | 75.39 |
| 10 | 685,279 | 1,385,975 | 2,071,254 | 40.73 | 59.27 | 80.82 |
| **AVE.** | 472,550 | 1,500,708 | 1,973,258 | 49.84 | 50.16 | 77.82 |
| **Range** | < 16 % | < 4 % | < 5 % | < 12 % | < 9 % | < 2 % |

### D.3.1 Increasing the Fleet Size with 25%

The fleet size was increased with 25% and seven runs executed with the maximum service level algorithm. The table below summarises the result of the weekly parameters.

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 0 | 4,355,772 | 4,355,772 | 100 | n/a | 23.65 |
| 2 | 0 | 4,343,867 | 4,343,867 | 100 | n/a | 24.53 |
| 3 | 0 | 4,331,804 | 4,331,804 | 100 | n/a | 25.32 |
| 4 | 0 | 4,354,437 | 4,354,437 | 100 | n/a | 25.24 |
| 5 | 0 | 4,414,696 | 4,414,696 | 100 | n/a | 25.71 |
| 6 | 0 | 4,410,864 | 4,410,864 | 100 | n/a | 24.95 |
| 7 | 0 | 4,363,073 | 4,363,073 | 100 | n/a | 25.31 |
| **Averages** | | **4,367,787** | **4,367,787** | **100** | **n/a** | **24.96** |
| **95% Conf. half width** | | **< 1 %** | **< 1 %** | **n/a** | **n/a** | **< 3 %** |

The MTC ($\delta$=10) algorithm was also used to schedule the larger fleet and the following table summarises the result of the weekly parameters:

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 1,005,180 | 1,419,559 | 2,424,739 | 44.02 | 55.98 | 82.02 |
| 2 | 766,098 | 1,538,531 | 2,304,629 | 52.43 | 47.57 | 76.66 |
| 3 | 675,843 | 1,652,351 | 2,328,194 | 54.56 | 45.44 | 73.63 |
| 4 | 692,950 | 1,660,596 | 2,353,546 | 55.20 | 44.80 | 75.77 |
| 5 | 806,967 | 1,400,383 | 2,207,351 | 53.05 | 46.95 | 82.01 |
| 6 | 728,192 | 1,667,850 | 2,396,042 | 59.41 | 40.59 | 80.90 |
| 7 | 1,095,383 | 1,450,827 | 2,546,210 | 49.66 | 50.34 | 80.26 |
| 8 | 957,551 | 1,553,099 | 2,510,650 | 50.91 | 49.09 | 76.71 |
| 9 | 717,797 | 1,551,405 | 2,269,202 | 50.49 | 49.51 | 78.05 |
| 10 | 955,044 | 1,473,791 | 2,428,835 | 45.17 | 54.83 | 79.82 |
| **AVE.** | 840,101 | 1,536,839 | 2,376,940 | 51.49 | 48.51 | 78.58 |
| **Range** | < 12 % | < 5 % | < 3 % | < 8 % | < 7 % | < 3 % |

## D.4. STUDY IN ADDING A DEPOT

After adding a depot, 10 replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the maximum service level heuristic:

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 0.00 | 2,414,186 | 2,414,186 | 100.00 | n/a | 40.50 |
| 2 | 0.00 | 2,397,072 | 2,397,072 | 100.00 | n/a | 39.49 |
| 3 | 0.00 | 2,493,911 | 2,493,911 | 100.00 | n/a | 37.32 |
| 4 | 0.00 | 2,418,302 | 2,418,302 | 100.00 | n/a | 40.19 |
| 5 | 0.00 | 2,462,597 | 2,462,597 | 100.00 | n/a | 40.76 |
| 6 | 0.00 | 2,413,988 | 2,413,988 | 100.00 | n/a | 39.82 |
| 7 | 0.00 | 2,454,439 | 2,454,439 | 100.00 | n/a | 38.93 |
| 8 | 0.00 | 2,430,514 | 2,430,514 | 100.00 | n/a | 40.77 |
| 9 | 0.00 | 2,436,245 | 2,436,245 | 100.00 | n/a | 38.62 |
| 10 | 0.00 | 2,415,293 | 2,415,293 | 100.00 | n/a | 38.75 |
| **AVE.** | 0.00 | 2,433,655 | 2,433,655 | 100.00 | n/a | 39.52 |
| **Range** | n/a | < 1 % | < 1 % | n/a | n/a | < 3 % |

After adding a depot, 10 replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the minimum total cost heuristic (with $\delta = 600$):

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | 114,527 | 1,786,155 | 1,900,682 | 81.44 | 18.56 | 66.84 |
| 2 | 122,088 | 1,947,105 | 2,069,193 | 81.50 | 18.50 | 59.55 |
| 3 | 106,574 | 1,965,428 | 2,072,002 | 84.23 | 15.77 | 60.43 |
| 4 | 142,624 | 2,030,816 | 2,173,440 | 79.20 | 20.80 | 57.81 |
| 5 | 157,660 | 1,935,997 | 2,093,657 | 75.43 | 24.57 | 63.87 |
| 6 | 87,568 | 2,015,555 | 2,103,124 | 81.04 | 18.96 | 58.62 |
| 7 | 81,138 | 2,303,732 | 2,384,870 | 87.48 | 12.52 | 55.28 |
| 8 | 94,634 | 1,904,459 | 1,999,093 | 85.97 | 14.03 | 58.35 |
| 9 | 125,200 | 1,931,727 | 2,056,927 | 83.59 | 16.41 | 64.43 |
| 10 | 103,284 | 2,030,645 | 2,133,929 | 83.90 | 16.10 | 56.90 |
| **AVE.** | 113,530 | 1,985,162 | 2,098,692 | 82.38 | 17.62 | 60.21 |
| **Range** | < 17 % | < 5 % | < 5 % | < 3 % | < 16 % | < 5 % |

## D.5. STUDY IN ACCEPTING A TENDER

After adding a tender as described in chapter 6, 10 replications were executed beyond the warm-up period and the weekly parameters determined. The table below summarises the result for the maximum service level heuristic:

| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | *10,734* | *3,322,214* | *3,332,948* | *95.43* | *4.57* | *48.10* |
| 2 | 2,863 | 3,246,946 | 3,249,809 | 97.87 | 2.13 | 48.53 |
| 3 | 18,692 | 3,305,341 | 3,324,034 | 93.44 | 6.56 | 46.18 |
| 4 | 4,550 | 3,316,065 | 3,320,615 | 97.02 | 2.98 | 47.33 |
| 5 | 26,452 | 3,194,547 | 3,221,000 | 92.33 | 7.67 | 49.93 |
| 6 | 10,676 | 3,249,495 | 3,260,171 | 93.27 | 6.73 | 47.43 |
| 7 | 7,450 | 3,362,515 | 3,369,965 | 96.06 | 3.94 | 50.21 |
| 8 | 5,209 | 3,295,110 | 3,300,319 | 96.93 | 3.07 | 45.68 |
| 9 | 197 | 3,454,862 | 3,455,059 | 99.70 | 0.30 | 47.11 |
| 10 | 5,431 | 3,228,176 | 3,233,608 | 97.39 | 2.61 | 48.82 |
| **AVE.** | 9,225 | 3,297,527 | 3,306,753 | 95.94 | 4.06 | 47.93 |
| **Range** | n/a | 1.66 | 1.57 | 1.71 | n/a | 2.17 |

Note: The container penalty cost and average container late vary too much to create sensible confidence intervals. The container penalty cost is small and does not have a significant effect on the total cost.

Ten replications were also executed beyond the warm-up period for this scenario using the MTC () algorithm. The table below summarises the result:

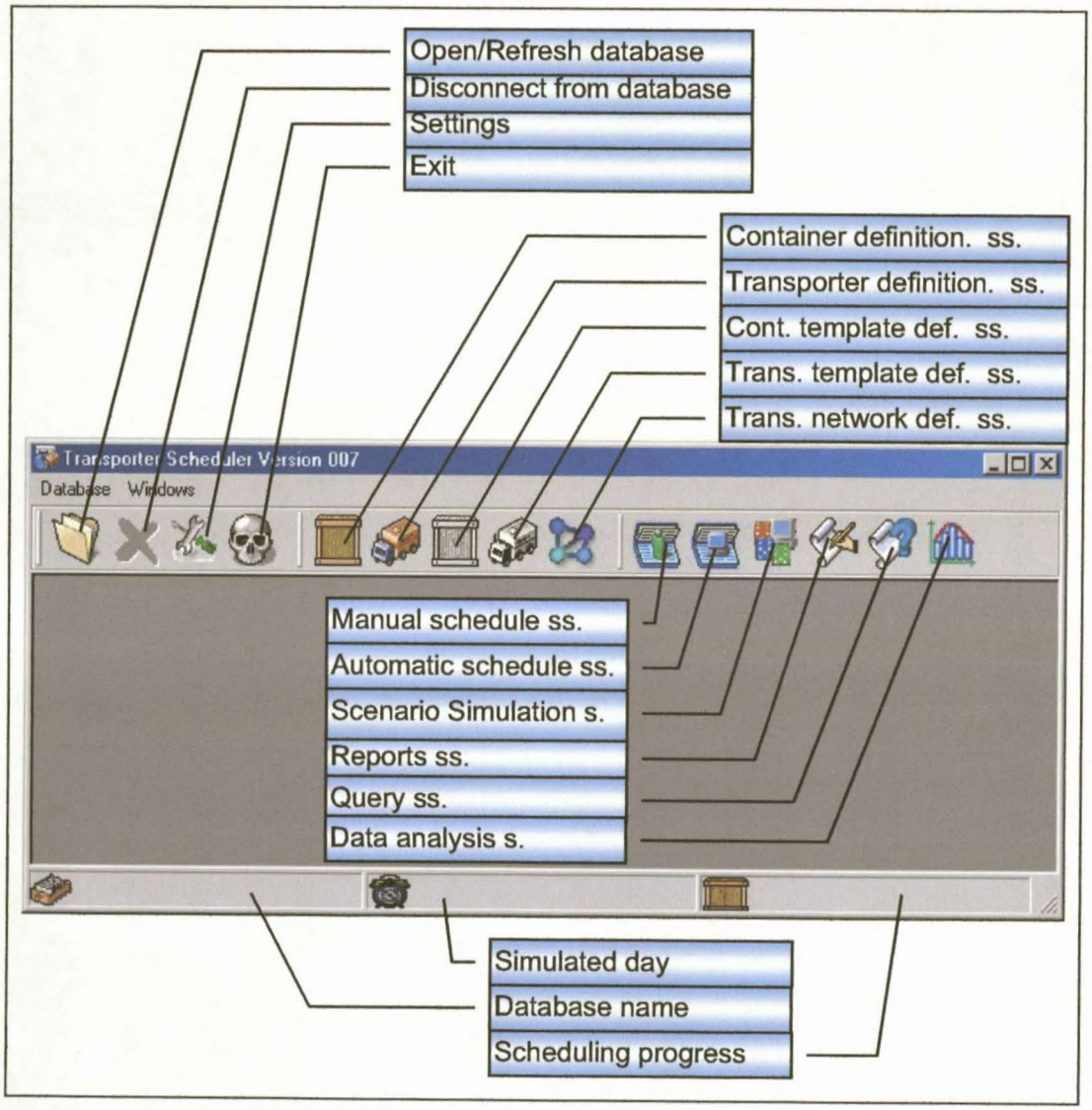
| *Run* | *Penalty Cost* | *Running Cost* | *Total Cost* | *Service Level [%]* | *Ave. Late Time [h]* | *Transporter Efficiency* |
|---|---|---|---|---|---|---|
| 1 | *215,669* | *3,330,192* | *3,545,861* | *72.57* | *27.43* | *53.00* |
| 2 | 215,866 | 3,243,665 | 3,459,531 | 73.32 | 26.68 | 54.96 |
| 3 | 170,198 | 3,519,007 | 3,689,205 | 76.31 | 23.69 | 54.70 |
| 4 | 198,013 | 3,395,846 | 3,593,859 | 71.35 | 28.65 | 51.50 |
| 5 | 218,940 | 3,500,972 | 3,719,912 | 73.00 | 27.00 | 53.41 |
| 6 | 233,096 | 3,392,104 | 3,625,200 | 74.70 | 25.30 | 53.88 |
| 7 | 215,728 | 3,411,595 | 3,627,323 | 74.64 | 25.36 | 52.97 |
| 8 | 214,553 | 3,375,804 | 3,590,357 | 73.07 | 26.93 | 52.00 |
| 9 | 198,844 | 3,369,353 | 3,568,197 | 77.00 | 23.00 | 53.96 |
| 10 | 200,012 | 3,262,143 | 3,462,155 | 76.19 | 23.81 | 54.71 |
| **AVE.** | 208,092 | 3,380,068 | 3,588,160 | 74.22 | 25.79 | 53.51 |
| **Range** | < 7 % | < 2 % | < 2 % | < 2 % | < 6 % | < 2 % |

# Appendix E User's Guide

An outline (with screenshots) of the program structure follows below.

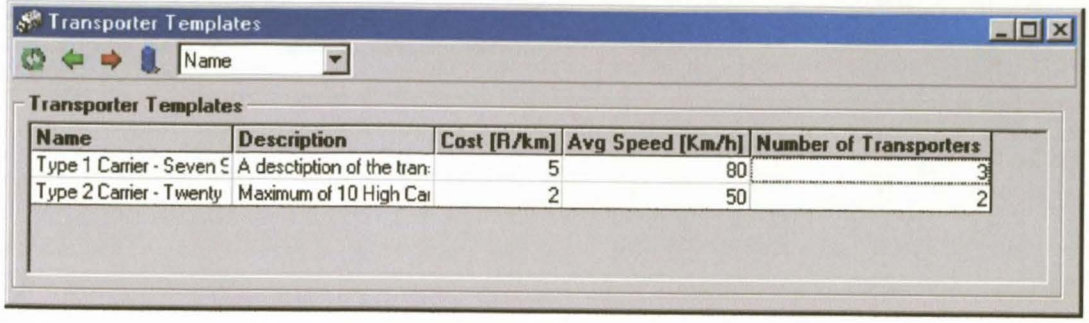
## E.0. MAIN PROGRAM STRUCTURE

The diagram below illustrates the main window and toolbar functionality.

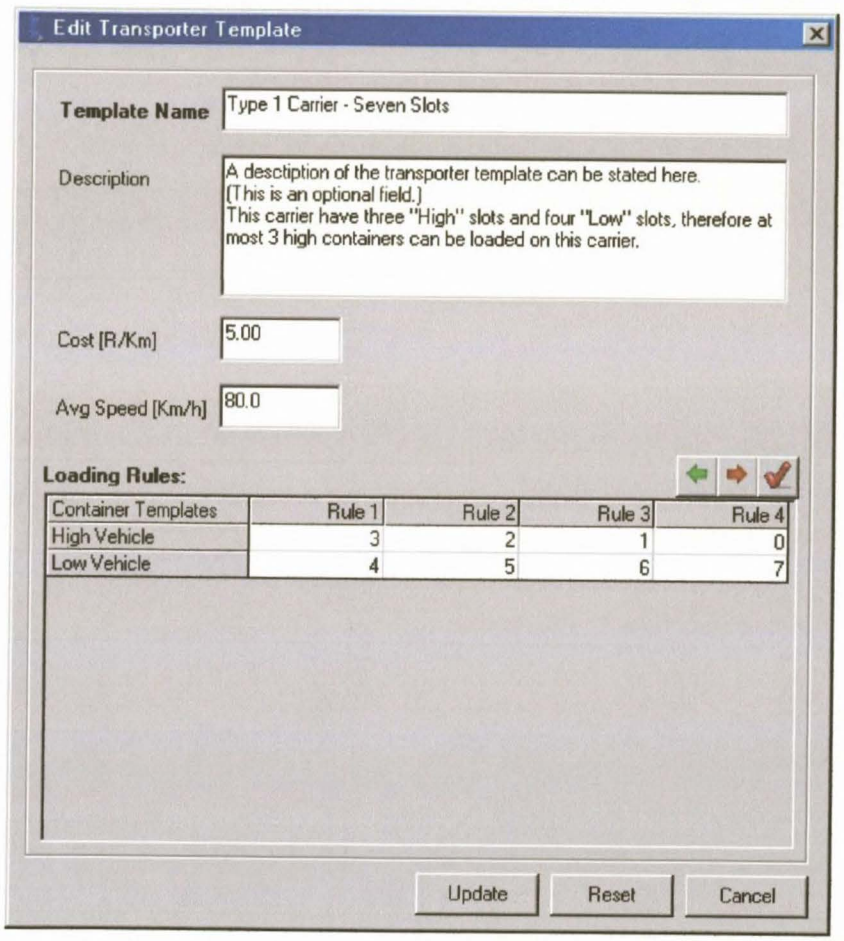


## E.1. PROBLEM DEFINITION STRUCTURE

### E.1.1. Transporter template definition sub-structure

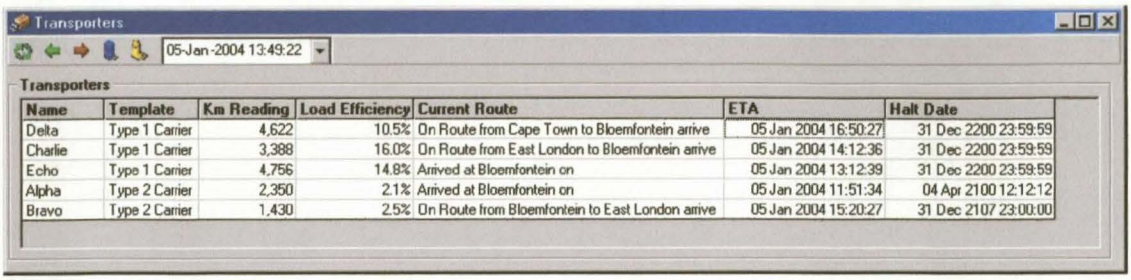



The screenshot below illustrates how a transporter template can be added.




Note: A loading capacity restriction can be checked for validity by clicking the "✓" button. (The loading capacity restriction is automatically checked before it is updated.)

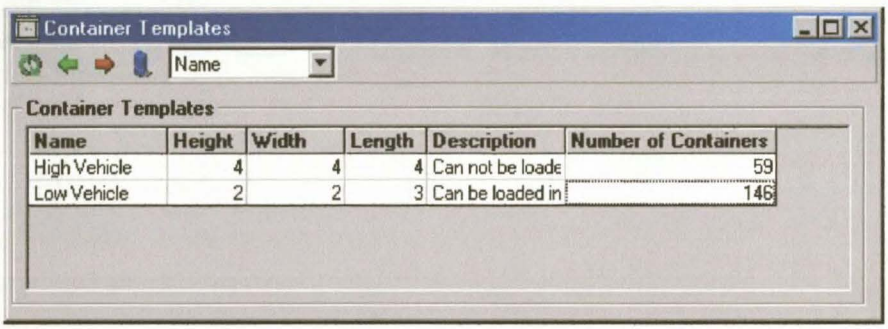
### E.1.2. Transporter definition sub-structure




| Name | Template | Km Reading | Load Efficiency | Current Route | ETA | Halt Date |
|---|---|---|---|---|---|---|
| Delta | Type 1 Carrier | 4,622 | 10.5% | On Route from Cape Town to Bloemfontein arrive | 05 Jan 2004 16:50:27 | 31 Dec 2200 23:59:59 |
| Charlie | Type 1 Carrier | 3,388 | 16.0% | On Route from East London to Bloemfontein arrive | 05 Jan 2004 14:12:36 | 31 Dec 2200 23:59:59 |
| Echo | Type 1 Carrier | 4,756 | 14.8% | Arrived at Bloemfontein on | 05 Jan 2004 13:12:39 | 31 Dec 2200 23:59:59 |
| Alpha | Type 2 Carrier | 2,350 | 2.1% | Arrived at Bloemfontein on | 05 Jan 2004 11:51:34 | 04 Apr 2100 12:12:12 |
| Bravo | Type 2 Carrier | 1,430 | 2.5% | On Route from Bloemfontein to East London arrive | 05 Jan 2004 15:20:27 | 31 Dec 2107 23:00:00 |

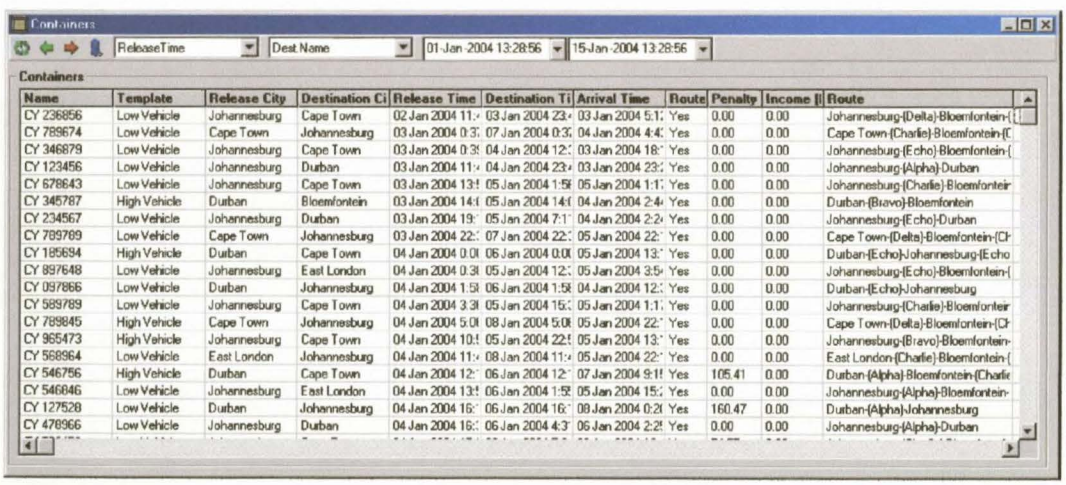
Note: Detailed information of routes of a transporter can be viewed by clicking the "i" button.

### E.1.3. Container template definition sub-structure



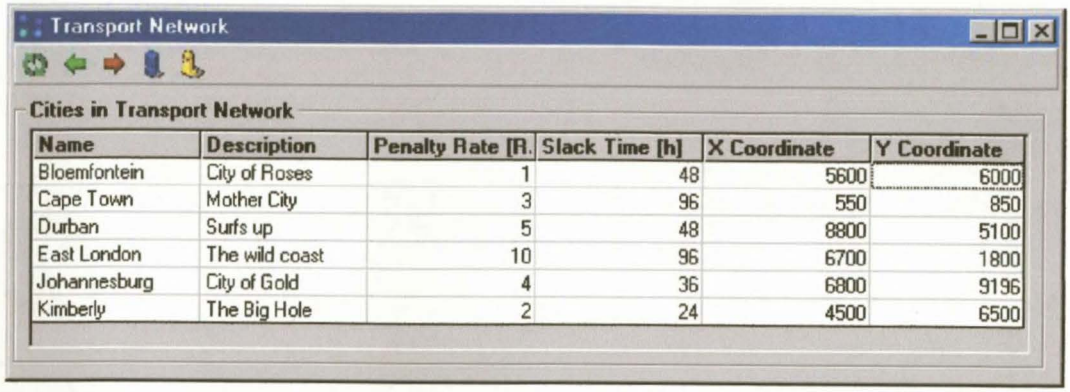

| Name | Height | Width | Length | Description | Number of Containers |
|---|---|---|---|---|---|
| High Vehicle | 4 | 4 | 4 | Can not be loade | 59 |
| Low Vehicle | 2 | 2 | 3 | Can be loaded in | 146 |

### E.1.4. Container Definition sub-structure



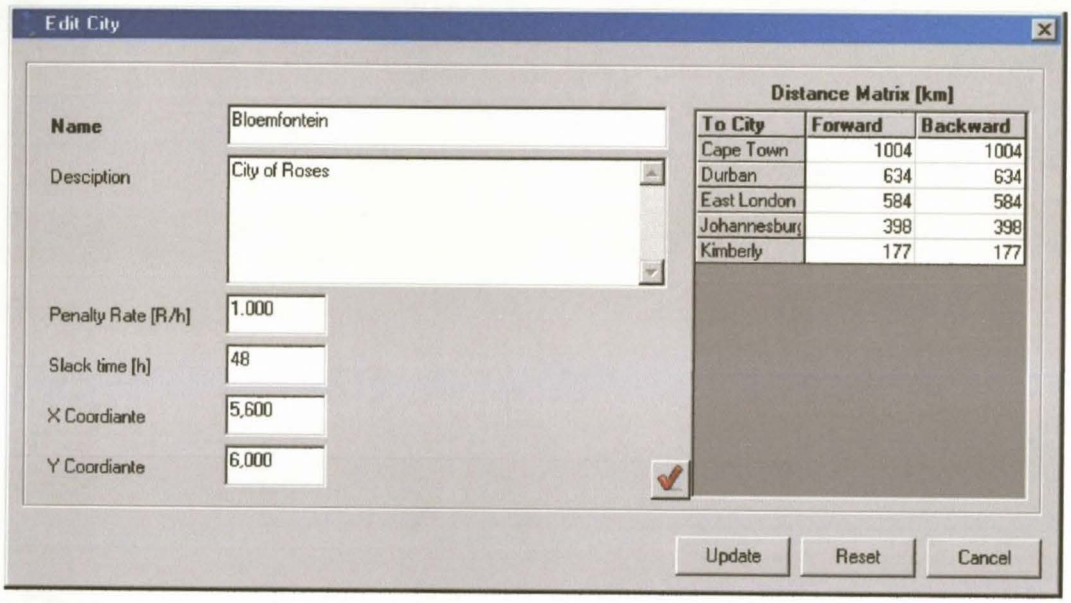

| Name | Template | Release City | Destination Ci | Release Time | Destination Ti | Arrival Time | Route | Penalty | Income [I | Route |
|---|---|---|---|---|---|---|---|---|---|---|
| CY 236856 | Low Vehicle | Johannesburg | Cape Town | 02 Jan 2004 11: | 03 Jan 2004 23: | 03 Jan 2004 5:1 | Yes | 0.00 | 0.00 | Johannesburg-(Delta)-Bloemfontein-( |
| CY 789674 | Low Vehicle | Cape Town | Johannesburg | 03 Jan 2004 0:3 | 07 Jan 2004 0:3 | 04 Jan 2004 4:4 | Yes | 0.00 | 0.00 | Cape Town-(Charlie)-Bloemfontein-( |
| CY 346879 | Low Vehicle | Johannesburg | Cape Town | 03 Jan 2004 0:3 | 04 Jan 2004 12: | 03 Jan 2004 18: | Yes | 0.00 | 0.00 | Johannesburg-(Echo)-Bloemfontein-( |
| CY 123456 | Low Vehicle | Johannesburg | Durban | 03 Jan 2004 11: | 04 Jan 2004 23: | 03 Jan 2004 23: | Yes | 0.00 | 0.00 | Johannesburg-(Alpha)-Durban |
| CY 678643 | Low Vehicle | Johannesburg | Cape Town | 03 Jan 2004 13: | 05 Jan 2004 1:5 | 05 Jan 2004 1:1 | Yes | 0.00 | 0.00 | Johannesburg-(Charlie)-Bloemfontein |
| CY 345787 | High Vehicle | Durban | Bloemfontein | 03 Jan 2004 14: | 05 Jan 2004 14: | 04 Jan 2004 2:4 | Yes | 0.00 | 0.00 | Durban-(Bravo)-Bloemfontein |
| CY 234567 | Low Vehicle | Johannesburg | Durban | 03 Jan 2004 19: | 05 Jan 2004 7:1 | 04 Jan 2004 2:2 | Yes | 0.00 | 0.00 | Johannesburg-(Echo)-Durban |
| CY 789789 | Low Vehicle | Cape Town | Johannesburg | 03 Jan 2004 22: | 07 Jan 2004 22: | 05 Jan 2004 22: | Yes | 0.00 | 0.00 | Cape Town-(Delta)-Bloemfontein-(Ch |
| CY 185694 | High Vehicle | Durban | Cape Town | 04 Jan 2004 0:0 | 06 Jan 2004 0:0 | 05 Jan 2004 13: | Yes | 0.00 | 0.00 | Durban-(Echo)-Johannesburg-(Echo |
| CY 897648 | Low Vehicle | Johannesburg | East London | 04 Jan 2004 0:3 | 05 Jan 2004 12: | 05 Jan 2004 3:5 | Yes | 0.00 | 0.00 | Johannesburg-(Echo)-Bloemfontein-( |
| CY 097866 | Low Vehicle | Durban | Johannesburg | 04 Jan 2004 1:5 | 06 Jan 2004 1:5 | 04 Jan 2004 12: | Yes | 0.00 | 0.00 | Durban-(Echo)-Johannesburg |
| CY 589789 | Low Vehicle | Johannesburg | Cape Town | 04 Jan 2004 3:3 | 05 Jan 2004 15: | 05 Jan 2004 1:1 | Yes | 0.00 | 0.00 | Johannesburg-(Charlie)-Bloemfontein |
| CY 789845 | High Vehicle | Cape Town | Johannesburg | 04 Jan 2004 5:0 | 08 Jan 2004 5:0 | 05 Jan 2004 22: | Yes | 0.00 | 0.00 | Cape Town-(Delta)-Bloemfontein-(Ch |
| CY 965473 | High Vehicle | Johannesburg | Cape Town | 04 Jan 2004 10: | 05 Jan 2004 22: | 05 Jan 2004 13: | Yes | 0.00 | 0.00 | Johannesburg-(Bravo)-Bloemfontein- |
| CY 568964 | Low Vehicle | East London | Johannesburg | 04 Jan 2004 11: | 08 Jan 2004 11: | 05 Jan 2004 22: | Yes | 0.00 | 0.00 | East London-(Charlie)-Bloemfontein-( |
| CY 546756 | High Vehicle | Durban | Cape Town | 04 Jan 2004 12: | 06 Jan 2004 12: | 07 Jan 2004 9:1 | Yes | 105.41 | 0.00 | Durban-(Alpha)-Bloemfontein-(Charlie |
| CY 546846 | Low Vehicle | Johannesburg | East London | 04 Jan 2004 13: | 06 Jan 2004 1:5 | 05 Jan 2004 15: | Yes | 0.00 | 0.00 | Johannesburg-(Alpha)-Bloemfontein- |
| CY 127528 | Low Vehicle | Durban | Johannesburg | 04 Jan 2004 16: | 06 Jan 2004 16: | 08 Jan 2004 0:2 | Yes | 160.47 | 0.00 | Durban-(Alpha)-Johannesburg |
| CY 478966 | Low Vehicle | Johannesburg | Durban | 04 Jan 2004 16: | 06 Jan 2004 4:3 | 06 Jan 2004 2:2 | Yes | 0.00 | 0.00 | Johannesburg-(Alpha)-Durban |

### E.1.5. Transportation network definition sub-structure



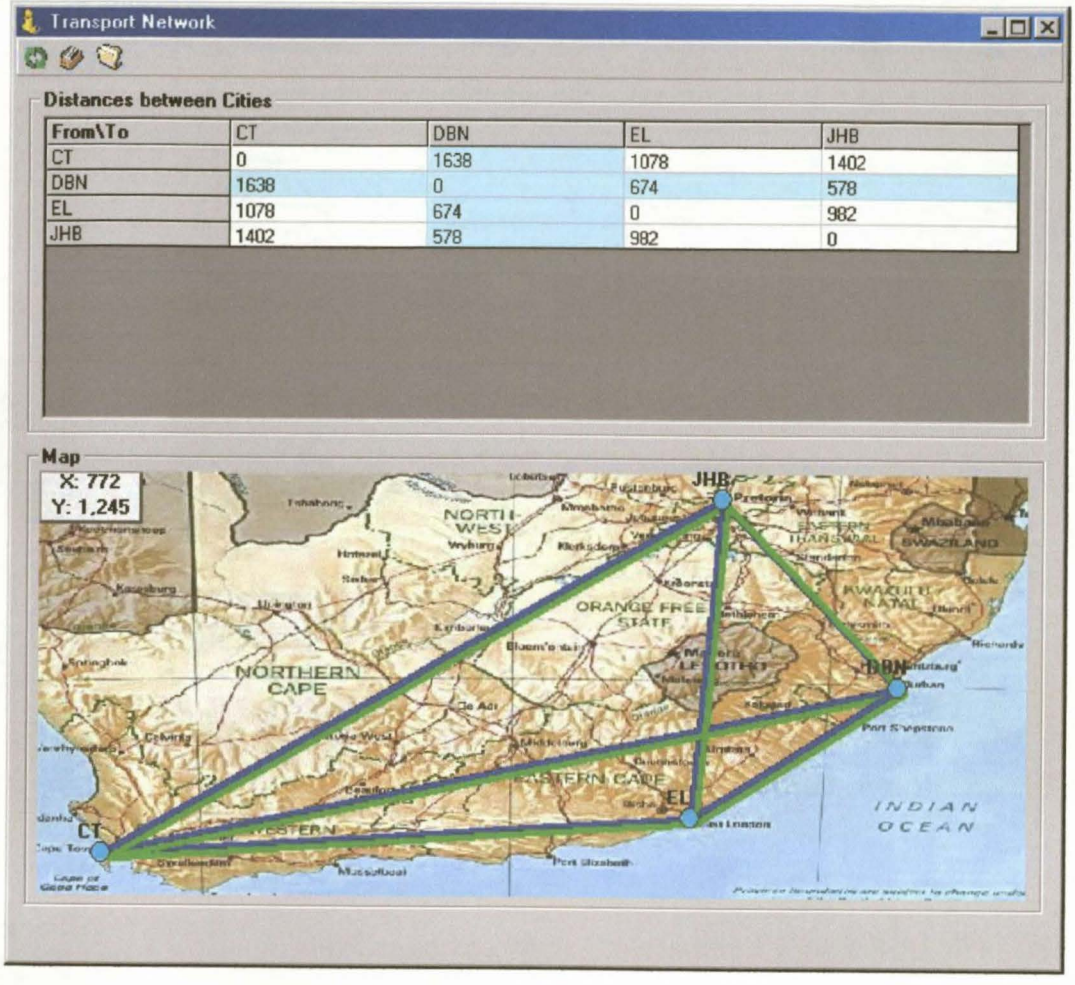

| Name | Description | Penalty Rate [R. | Slack Time [h] | X Coordinate | Y Coordinate |
|---|---|---|---|---|---|
| Bloemfontein | City of Roses | 1 | 48 | 5600 | 6000 |
| Cape Town | Mother City | 3 | 96 | 550 | 850 |
| Durban | Surfs up | 5 | 48 | 8800 | 5100 |
| East London | The wild coast | 10 | 96 | 6700 | 1800 |
| Johannesburg | City of Gold | 4 | 36 | 6800 | 9196 |
| Kimberly | The Big Hole | 2 | 24 | 4500 | 6500 |

The screenshot below illustrates how a vertex can be added.




Note: the graph can be checked for connectedness by clicking the "✓" button.

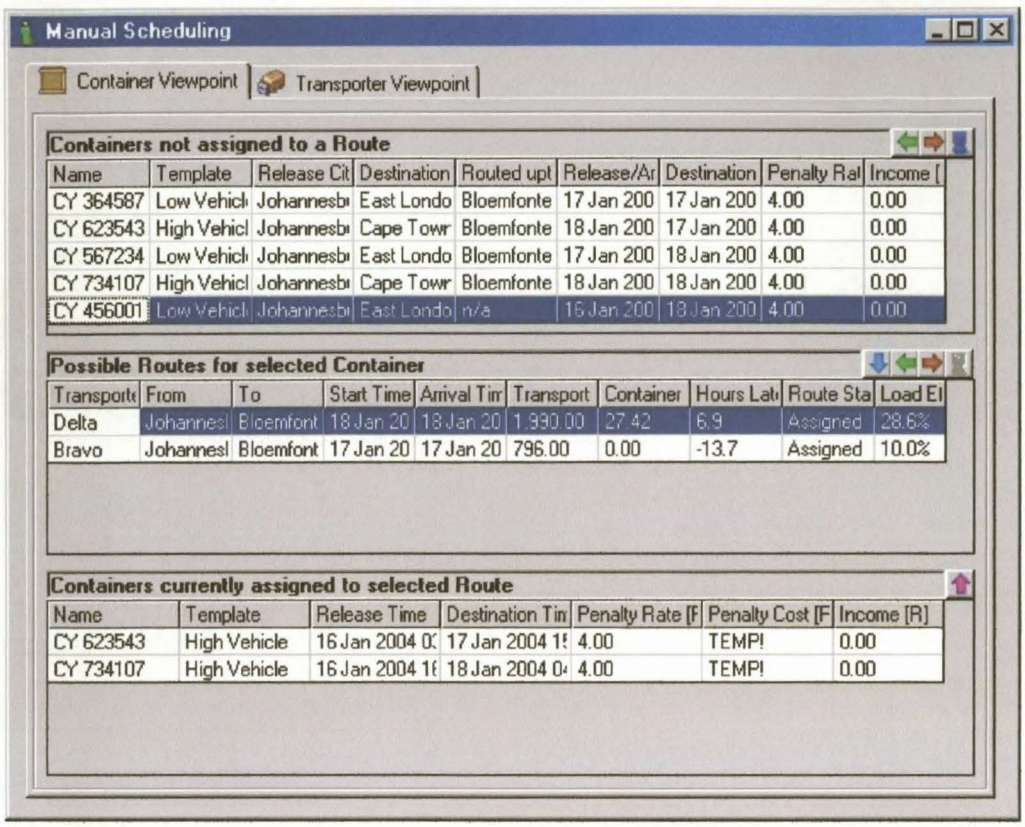
The transportation network can be displayed as shown below.



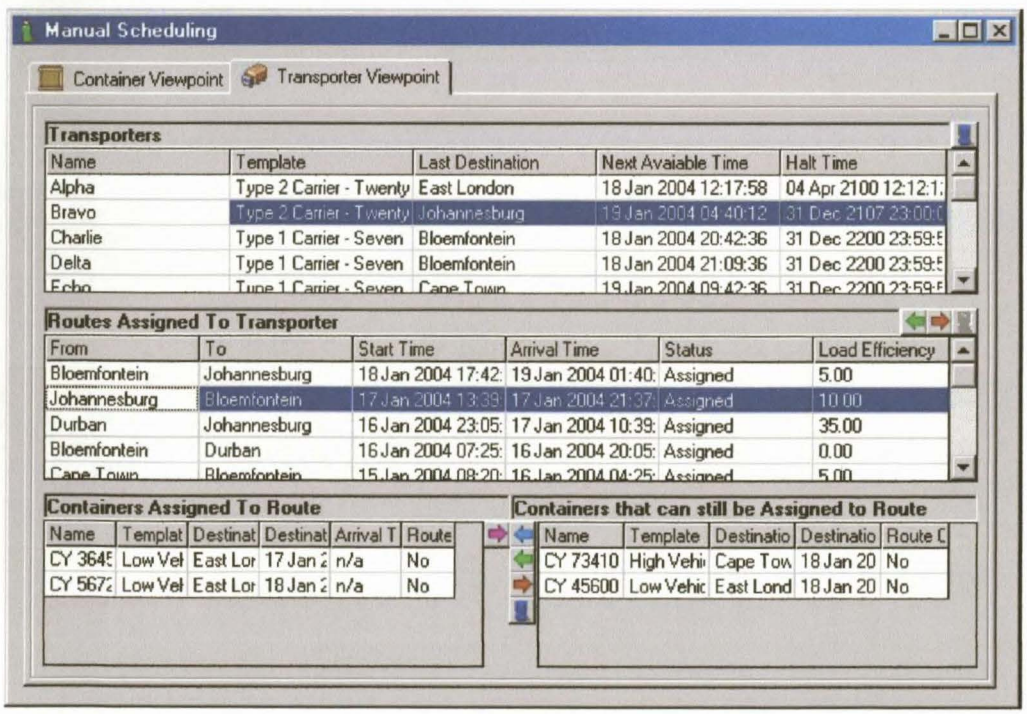
## E.2. THE SOLUTION DEVELOPMENT STRUCTURE

### E.2.1. The manual scheduling sub structure

The container view point is shown below



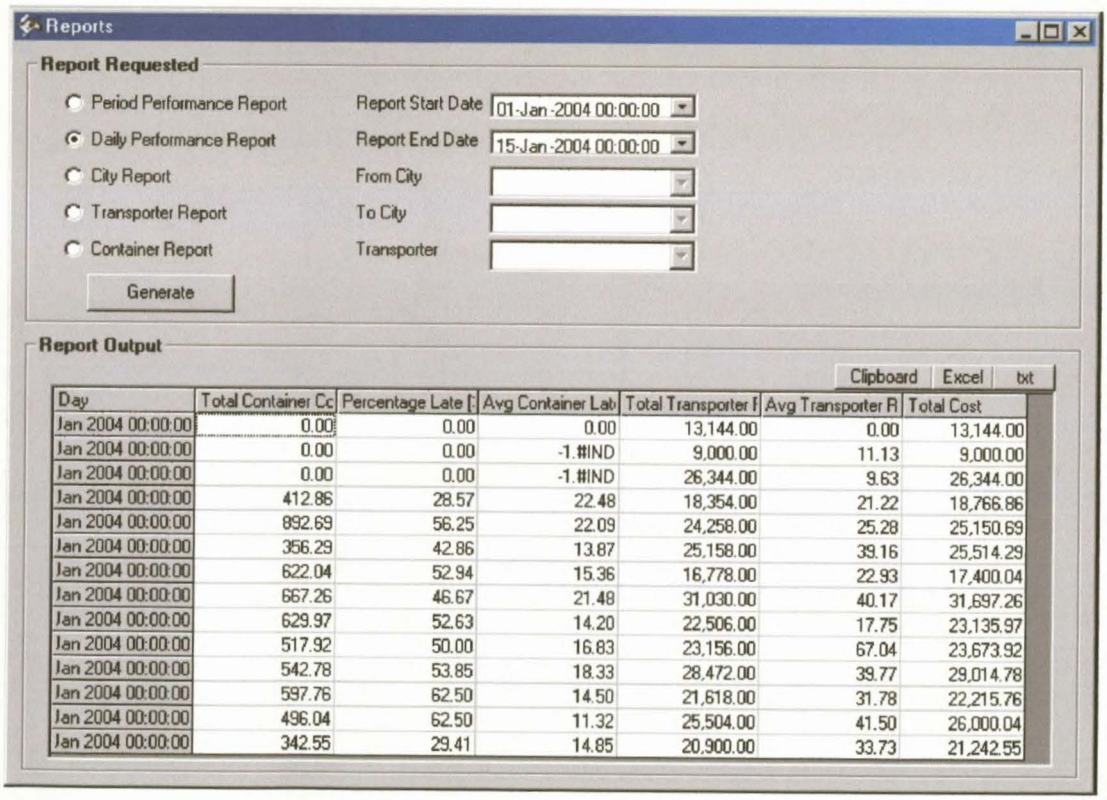

The transporter viewpoint is shown below



### E.2.2. The automatic scheduling sub-structure

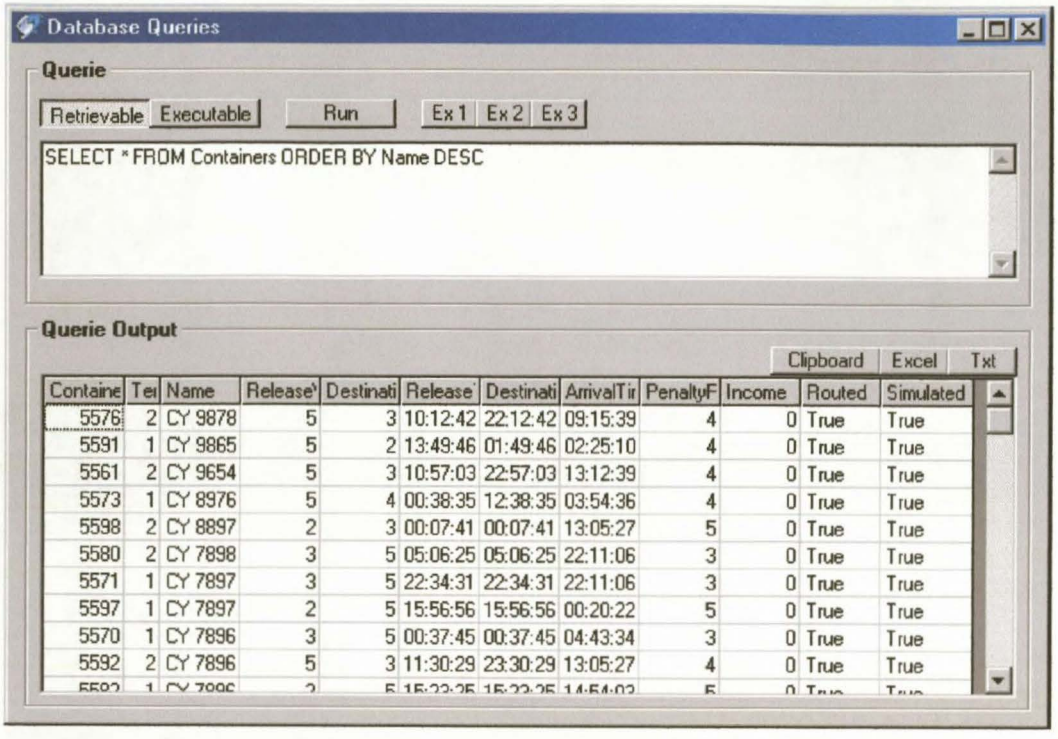

## E.3. THE SOLUTION EVALUATION STRUCTURE

### E.3.1. The reports sub-suture

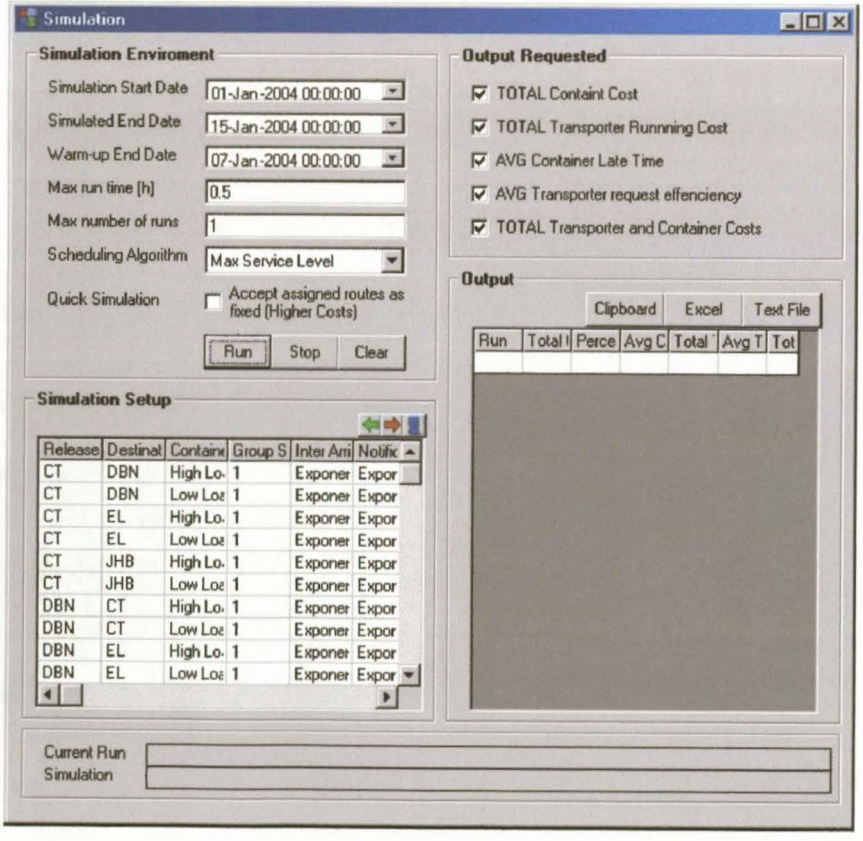



| Day | Total Container Cc | Percentage Late [: | Avg Container Lat | Total Transporter I | Avg Transporter R | Total Cost |
|---|---|---|---|---|---|---|
| Jan 2004 00:00:00 | 0.00 | 0.00 | 0.00 | 13,144.00 | 0.00 | 13,144.00 |
| Jan 2004 00:00:00 | 0.00 | 0.00 | -1.#IND | 9,000.00 | 11.13 | 9,000.00 |
| Jan 2004 00:00:00 | 0.00 | 0.00 | -1.#IND | 26,344.00 | 9.63 | 26,344.00 |
| Jan 2004 00:00:00 | 412.86 | 28.57 | 22.48 | 18,354.00 | 21.22 | 18,766.86 |
| Jan 2004 00:00:00 | 892.69 | 56.25 | 22.09 | 24,258.00 | 25.28 | 25,150.69 |
| Jan 2004 00:00:00 | 356.29 | 42.86 | 13.87 | 25,158.00 | 39.16 | 25,514.29 |
| Jan 2004 00:00:00 | 622.04 | 52.94 | 15.36 | 16,778.00 | 22.93 | 17,400.04 |
| Jan 2004 00:00:00 | 667.26 | 46.67 | 21.48 | 31,030.00 | 40.17 | 31,697.26 |
| Jan 2004 00:00:00 | 629.97 | 52.63 | 14.20 | 22,506.00 | 17.75 | 23,135.97 |
| Jan 2004 00:00:00 | 517.92 | 50.00 | 16.83 | 23,156.00 | 67.04 | 23,673.92 |
| Jan 2004 00:00:00 | 542.78 | 53.85 | 18.33 | 28,472.00 | 39.77 | 29,014.78 |
| Jan 2004 00:00:00 | 597.76 | 62.50 | 14.50 | 21,618.00 | 31.78 | 22,215.76 |
| Jan 2004 00:00:00 | 496.04 | 62.50 | 11.32 | 25,504.00 | 41.50 | 26,000.04 |
| Jan 2004 00:00:00 | 342.55 | 29.41 | 14.85 | 20,900.00 | 33.73 | 21,242.55 |

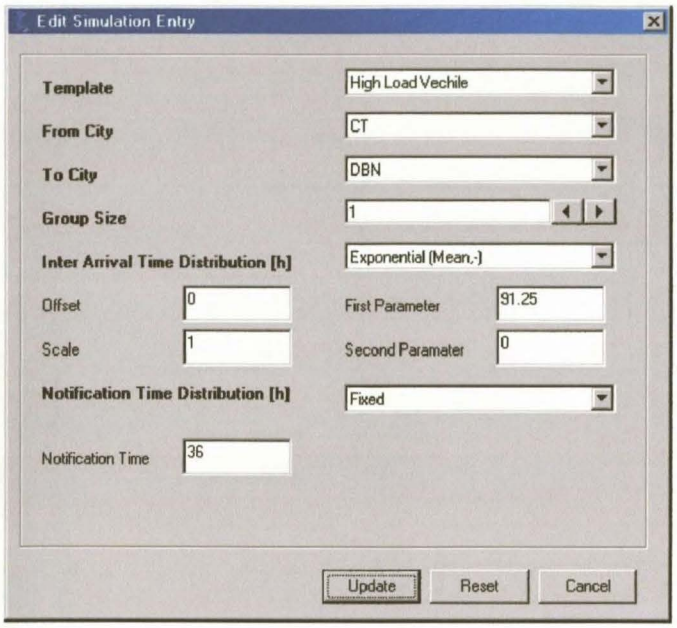
### E.3.2. The query sub-structure




## E.4. THE SCENARIO SIMULATION STRUCTURE



The screenshot below illustrates how a container entering pattern can be edited.




## E.5. THE DATA ANALYSIS STRUCTURE