The Genetic Algorithm for solving the Dial-a-Ride Problem

Kristin Berg Bergvinsdottir

Kgs. Lyngby

Thesis-2004-37

IMM

Trykt af IMM, DTU

Preface

This theses is submitted in partial fulfillment of the requirements of the degree: *Master of Science in Engineering.* The project was prepared by Kristin Berg Bergvinsdottir during the period October 2003 to May 2004 at the Operations Research section at the department of Informatics and Mathematical Modelling (IMM), Technical University of Denmark (DTU). The work was supervised by lector Jesper Larsen.

I would like to thank my supervisor for his assistance throughout the project. He assisted me on all the stages of the work and was always prepared to assist. He was also full of good ideas and inspiration.

I wold also like to thank René Munk Jørgensen, who acted as an assistant supervisor. He had a great inside knowledge about the dial-a-ride problem and helped me to get a much better understanding of the practical side of the problem. He was very supporting and encouraging throughout the entire working period.

Special thanks go to my family for their support, especially during the last phases of the project.

Kgs. Lyngby, May 2004,

Kristín Berg Bergvinsdóttir
s001341

Abstract in English

In this project the genetic algorithm (GA) is used to solve the dial-a-ride problem (DARP). In a dial-a-ride system customers are to be picked up and delivered within given time windows by a transportation vehicle. The aim is to minimize transportation cost and maximize the customer service level while satisfying the constraints.

The approach used to solve the DARP is the classical cluster-first, route-second strategy. That is, the customers are assigned to vehicles using a genetic algorithm and then a routing heuristic algorithm developed by Baugh et al. [2] is used to make a route for each vehicle. The solution method is implemented in JAVA and tested on several randomly generated test problems. The test problems are taken from Cordeau and Laporte [4]. Several improvement strategies to the initial solution method are proposed and the results from the best strategy are compared to the results obtained by Cordeau and Laporte.

Keywords:

Dial-a-Ride problem, meta-heuristic, combinatorial optimization, evolutionary algorithms, genetic algorithm, cluster-first, route-second, space-time nearest neighbour heuristic.

Resumé på dansk

I dette projekt er den genetiske algoritme (GA) brugt for at løse Dial-a-Ride Problemet (DARP). I DARP skal kunder hentes og afleveres af et transportmiddel indenfor givne tidsvinduer. Formålet er at minimere transport omkosninger og maximere kundeservice samtidigt med at holde begrænsningerne.

Metoden der er brugt til at løse DARP er den klassiske cluster-først, rute-næst strategi. I første omgang er den genetiske algoritme er brugt til at tildele kunder til transport midlerne og dernæst anvendes en rute algoritme lavet af Baugh et al. [2] til at lave ruten til hvert transport middel. Løsningsmetoden er implementeret i JAVA og testet med mange tilfældigt lavet test problemer. Test problemerne er lavet af Cordeau and Laporte [4]. Nogle forbedringsstrategier bliver præsenteret til den første løsningsmetode og resultaterne fra den bedste strategi bliver sammenlignet med resultaterne af Cordeau and Laporte.

Nøgleord:

Dial-a-Ride Problemet, meta-heuristik, combinatorial optimering, evolutionary algoritme, genetiske algoritme, cluster-først, rute-næst, areal-tid nærmeste nabo heuristik.

Contents

1 Introduction **9**

1.1 Examples of a Dial-a-Ride transportation system . . . 9

1.2 Purpose of the project . . . 12

1.3 Outline . . . 12

1.4 Abbreviations . . . 13

2 Dial-a-ride problem **15**

2.1 Characteristics of the dial-a-ride problem . . . 15

2.2 Customers with special needs . . . 16

2.3 Multi-objective optimization . . . 16

2.4 Static vs. dynamic dial-a-ride problem . . . 17

2.5 Inbound vs. outbound customers . . . 18

2.6 Problem formulation . . . 18

2.7 Notation in the Mathematical Model . . . 21

2.8 Mathematical model . . . 22

2.9 Objective function . . . 22

2.10 Constraints . . . 23

2.10.1 Depot constraints . . . 23

2.10.2 Routing constraints . . . 23

2.10.3 Precedence constraints . . . 23

2.10.4 Timing constraints . . . 24

2.10.5 Capacity constraints . . . 25

2.10.6 Linearization of constraints . . . 26

2.11 Related problems . . . 26

2.12 The Dial-a-ride problem is $\mathcal{NP}$-hard . . . 28

3 Previous Work **29**

3.1 Jaw et al. . . . 29

3.2 Baugh et al. . . . 30

3.3 Cordeau and Laporte . . . 32

3.4 Jih et al. . . . 33

3.5 Pereira et. al . . . 34

3.6 Comparison . . . 36

4 Solution method **39**

4.1 Choosing the solution method . . . 39

4.2 The Genetic Algorithm . . . 41

4.2.1 Original version of GA . . . 43

4.2.2 Other versions of GA . . . 43

4.2.3 Convergence . . . 44

4.2.4 Chromosome representation . . . 45

4.2.5 Schemata . . . 46

4.2.6 Population size . . . 47

4.2.7 Initial population . . . 48

4.2.8 Stopping criteria . . . 48

4.2.9 Fitness calculations . . . 48

4.2.10 Selection mechanism . . . 49

4.2.11 Modifying operators . . . 51

4.3 Modified space-time nearest neighbour heuristic . . . 55

5 Implementation **57**

5.1 Overview of the solution method . . . 57

5.2 Relaxation . . . 58

5.2.1 Effect of arrival time to destination . . . 60

5.3 The initial heuristic - GA1 . . . 61

5.3.1 Clustering using GA . . . 62

5.3.2 Routing strategy . . . 67

5.3.3 Overview of GA1 . . . 68

5.4 Improvements to GA1 . . . 69

6 Experimental results **73**

6.1 Data . . . 74

6.2 Testing and tuning the initial heuristic - GA1 . . . 76

6.2.1 Selecting values for the parameters in the GA . . . 76

6.3 Selecting values for the routing parameters in GA1 . . . 81

6.3.1 GA1 - Customers choice . . . 82

6.3.2 GA1 - Operators choice . . . 85

6.3.3 Weights chosen and convergence . . . 86

6.4 Improving the initial heuristic . . . 87

6.4.1 Size of generation equals population size - GA2 . . . 88

6.4.2 Number of duplicates reduced - GA3 . . . 88

6.4.3 Reducing the randomness in the heuristic - GA4 . . . 89

6.4.4 Heuristic run twice and total number of iterations constant - GA5 . 92

6.5 The best improvement . . . 95

6.6 Comparison with Cordeau and Laporte results . . . 95

6.7 Summary . . . 97

7 Conclusion **99**

Bibliography **101**

A JAVA source code **103**

Chapter 1

Introduction

In the dial-a-ride problem (DARP) customers give a transportation operator requests for transportation. A request consists of a specified pickup (origin) location and drop off (destination) location along with a desired departure or arrival time and the number of passengers to be transported. The problem is to determine the best routing schedule for the transporting vehicles, which minimizes overall transportation cost and yet maintains a high level of service to customers. The service level estimation is usually based on the ride times of the customers and deviations from desired departure or arrival times. It is very hard to combine these conflicting factors, small cost of transportation versus high service level, so a good compromise is what is aimed at.

1.1 Examples of a Dial-a-Ride transportation system

The dial-a-ride transportation system is used to describe a variety of transportation service systems. The simplest form of a dial-a-ride transportation system is the taxi transportation system. In the taxi system, a customer calls in with his request and is then transported directly from his pickup location to destination location. That is an example of a door-to-door transportation system. The aim of the dial-a-ride problem here, is to minimize the operators cost and maximize service to customers. The operators cost is decreased by minimizing the number of taxis standing by, waiting to service customers and the service level is increased by minimizing the waiting times of the customers before the taxi's arrival, since if the customer has to wait too long, he will turn his business elsewhere.

Another example of a dial-a-ride system is the specialized transportation, that is the transportation of for example children, disabled and elderly people. These specialized transports are usually provided by local authorities. The customers also call in their transportation requests, but here larger vehicles, such as mini busses off various capacities, are used to transport the passengers. In this kind of transportation it is also necessary to consider the different needs of the customers. Some customers require just a regular seat, others have to be seated in their wheelchair, while again others may have to lie down when being transported.

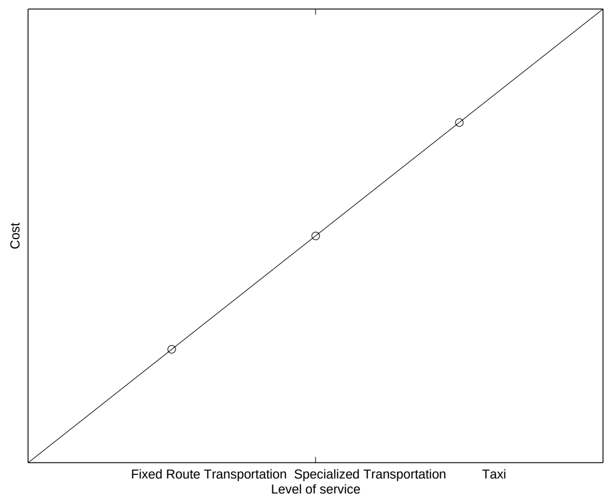

Figure 1.1: The three different modes of transportation cost more as the level of service increases. In this graph it is assumed that the cost increases linearly for fixed route, specialized and taxi transportation.

In order to decrease transportation costs it is necessary to organize the transportation in such a way, that different customers and their companions share a ride. That is, a customer might not be transported directly from his pickup location to his destination location but instead other customers could be dropped off or picked up in between. Therefore the dial-a-ride problem becomes more complex than what was the case in the taxi transportation system. There it was necessary to minimize the number of vehicles used for transportation but here it is also necessary to minimize the total distance and/or route duration for all the vehicles used in the transport.

As in the taxi system it is also necessary to consider the customers service levels, even though the customers usually cannot freely choose their transportation operator service, since it is most often a publicly provided service. But lack of punctuality due to late arrival by the transportation operator can cause problems and cost money in other organizations, which are perhaps also providing services to the customer. The unsatisfied customer will demand improvements and is sometimes entitled to that by rules or regulations or simply by moral law. The estimation of customers service level in this case is mainly based on deviations in arrival time and excess ride time, i.e. how much longer a customer has to sit in the bus than if it were a direct transport. Another factor to consider with regards to the service level is the amount of time a customer has to sit in a halting vehicle. The vehicle may have to halt in order to arrive at a correct time to collect or drop off a customer. Waiting times like this are tedious for the customers and cause dissatisfaction. Yet another factor that can be considered when looking at the service level is the shape of the route driven by the bus. That is, even though it might be optimal to drive in many loops crossing them-selves it does not look like a good plan for the customer or even the bus driver. Such routes do therefore not seem practical. Instead the usual flower shaped routes where different routes do not intersect, see figure 1.2, are used, even though they are more expensive. Other factors such as comfortability of vehicles, the manners of the bus driver and etc. are much harder to measure and will not be included in the dial-a-ride problem. These are examples of the many different factors that can influence the level of

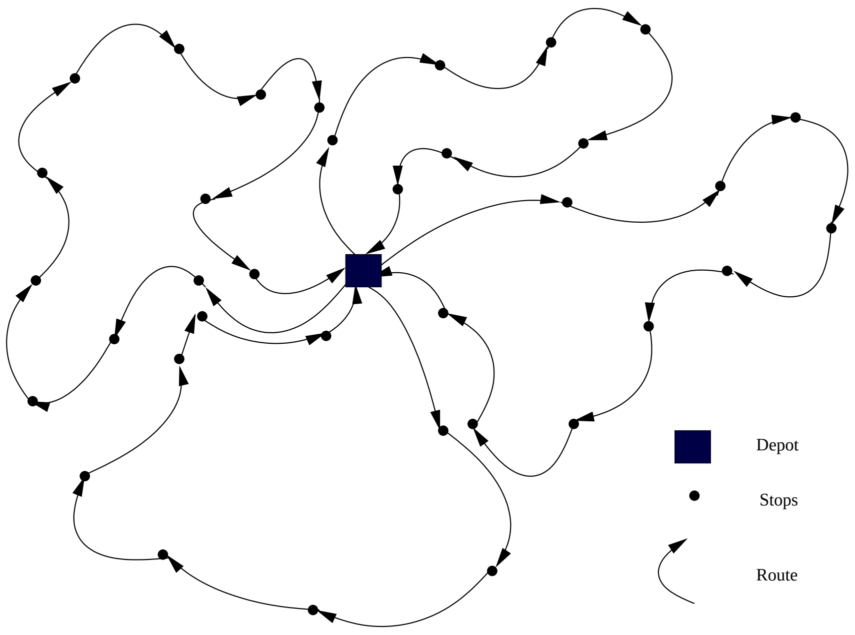

Figure 1.2: An example of flower shaped routes, i.e. where the routes do not intersect each other. Flower shaped routes are commonly encountered in practice.

service. Many other factors exist depending on the underlying problem. It varies which factors are included in the dial-a-ride problem. Decisions about which factors to include are based on the underlying problem.

Another example of a case, which can be described by a dial-a-ride transportation system, but which is not necessarily door-to-door transportation, is the public transportation in rural areas. Here the customers are picked up at fixed bus stops and the bus routes are also fixed. Furthermore the transportation fleet also consists of mini busses, possibly with different capacities, and the customers call in their transportation requests. The transportation operator then constructs a bus schedule accordingly, i.e. decides on the frequency and size of bus to drive a certain route.

The dial-a-ride transportation systems are very flexible and it is possible in a DAR system to combine both fixed routes transportation, with fixed stops and routes, and variable routes transportation, which depend on advance transportation requests from customers. For example in rural areas, where the bus drives a fixed route but can take a detour to drive an elderly, a disabled, a child or a special paying customer all the way to the door. In figure 1.1 it is shown an example of how the cost increases with the level of service provided. The highest service level is provided by taxi transportation but the taxi transportation service is also the most costly. It is most common to use a DAR system to describe the specialized transportation which lies in the middle of taxi and fixed routes transportation both in cost and level of service. In the figure it is assumed that the cost increases linearly with the level of service but that need not be the case.

The dial-a-ride transportation systems can also be used to describe transportation of for example animals and goods that are highly sensitive to their treatment during the transportation. The possibilities of usages are many and therefore it is very important to investigate the dial-a-ride problem in detail.

1.2 Purpose of the project

The work performed in this project is purely theoretical, i.e. the problem to be investigated is not based on a specific real-life problem. The problem will although be formulated based on realistic assumptions taken from a Danish transportation system.

The main goal of this project is to model and solve the dial-ride-problem using a solution method, which has not yet been used to solve the dial-a-ride problem. The solution method of choice in this project is cluster-first and route-second. The clustering will be solved using the genetic algorithm and the routing will be determined by a modified space-time nearest neighbour heuristic developed by Baugh et al. [2].

The solution method will be implemented in JAVA and tested using randomly created data sets generated by Cordeau and Laporte [4]. The results obtained in this project are compared with the results obtained by Cordeau and Laporte.

1.3 Outline

The remainder of this report will be divided into six chapters, which are described here briefly.

The dial-a-ride problem will be described in more detail in **chapter 2**, where an overview will be given of the most important related problems. A discussion of a multi-objective optimization, followed by a discussion of the difference between the static and the dynamic dial-a-ride problem and also the difference between an outbound and inbound customer. There after the problem that is considered in this work is formulated and the mathematical model for the DARP will be presented. There will be a description of the objective function as well as the constraints. Lastly the $\mathcal{NP}$-hardness of DARP will be addressed shortly.

In **chapter 3** a short introduction to the history of the research on the dial-a-ride problem is given. There will also be a description of some of the work that has previously been published about the dial-a-ride problem and related problems. The work described is work that has given inspiration and ideas to this project.

Chapter 4 gives a description of the solution method chosen to solve the DARP. The solution method is based on the cluster-first, route-second strategy. The clustering is the assignment of customers to vehicles, which is solved using the genetic algorithm. In the routing stage the customers already assigned to a vehicle are ordered, i.e. the actual route of the vehicle is constructed. The routing is solved using a heuristic adopted from Baugh et al. [2]. The chapter starts by explaining the choice of the solution method followed by a description of the genetic algorithm and the modified space time nearest neighbour heuristic.

In **chapter 5** the implementation of the solution method chosen for solving the dial-a-ride problem will be described. The problem is simplified through relaxation of constraints and solved in two stages, i.e. cluster-first, route-second. An initial heuristic is developed and several improvements proposed.

The experimental results are presented in **chapter 6**. The implemented algorithm will be tested on several randomly generated problem instances constructed by Cordeau and Laporte [4]. The chapter will start by an introduction to the data instances used. Then the some parameters in the genetic algorithm are tuned and the influence of the weights on the different parts of the objective function will be investigated. Next the initial algorithm will be tested on several instances. Thereafter some experiments to improve the algorithm are tested. Lastly the best strategy found will be tested further and compared with the results found by Cordeau and Laporte.

The final conclusions of this project will be given in **chapter 7**.

1.4 Abbreviations

Table 1.1: Table of abbreviations.

Abbreviation	Full text
CPU	Central Processing Unit
DAR	Dial-a-Ride
DARP	Dial-a-Ride problem
GA	Genetic Algorithm
LP	Linear Problem
MILP	Mixed-Integer Linear Problem
MTC	Montreal Transit Commission
$\mathcal{NP}$-hard	Non-deterministic Polynomial-time hard
OR	Operations Research
PDP	Pickup and Delivery Problem
TSP	Travelling Salesman Problem
TW	Time Windows
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls

Chapter 2

Dial-a-ride problem

In this chapter a more detailed description of the dial-a-ride problem is presented and related issues will be addressed. First the characteristics of the DARP are discussed and the multi-objective optimization is described. An introduction of the static and dynamic versions of the DARP and the difference between an outbound and inbound passenger will be given. Then a formulation of the specific problem used in the remainder of this report is given.

A basic mathematical model for the dial-a-ride problem with time windows is also presented along with a discussion of the various constraints in the problem and the objective function. The mathematical model that will be presented is very similar to the mathematical model presented by Jørgensen [11] with auditorial extensions some of which are taken from Baugh et al. [2]. The chapter ends by a description of related problems followed by a discussion of the difficulty in solving the dial-a-ride problem since it can be proven to be $\mathcal{NP}$-hard[1].

2.1 Characteristics of the dial-a-ride problem

The main characteristics of the static dial-a-ride problem with time windows will be described shortly in this section. These characteristics lay the foundation for formulating the mathematical model for the problem.

The objective of the dial-a-ride problem is to minimize total transportation costs and at the same time maximize the level of service provided to the customers. In this project it is assumed that the maximization of the level of service is to be equivalent to minimizing the unhappiness of the customers. The customers must be picked up or delivered within a given time interval. It is assumed that all customer requests are known in advance. It is required that each vehicle starts and ends in a depot but not necessarily the same depot. The customers must first be picked up and then dropped off by the same vehicle.

[1]In computational complexity theory, $\mathcal{NP}$-hard refers to the class of decision problems that contains all problems H such that for all decision problems L in $\mathcal{NP}$ there is a polynomial-time many-one reduction to H

The vehicles have a fixed capacity, which may not be exceeded at any time. There is also an upper limit on the route duration for each vehicle and ride time for each customer. The constraints on time windows, ride times, route duration and capacity of the vehicles can either be presented as soft or hard constraints. Hard constraints are constraints that cannot be violated while soft constraints can be violated but it adds to the cost. Which type of constraint is used, hard or soft, is governed by the underlying problem.

2.2 Customers with special needs

Since the dial-a-ride systems are often used to describe cases which involve the transportation of elderly and/or disabled persons this needs to be taken into account when making the model. The needs of these passengers are not the same as for other passengers. They may need assistance to get into/out of the vehicle, need special seats and so on. In order to get these factors into the model there is usually defined a service time associated with each stop, which accounts for the bus driver helping the passenger into the bus and secure him on the bus. The need for special seats can be incorporated into the model by specifying different capacities for each seat type on the vehicle and then keeping track of the load changes in each seat type. Another possibility is to define the vehicles with one capacity and then define the needs for special seats in number of regular seats. For example one passenger in a wheelchair needs two regular seats and a lying person needs four regular seats on the bus.

2.3 Multi-objective optimization

In the dial-a-ride problem the objective is to minimize total transportation cost and minimize the customer unhappiness. This kind of a optimization problem is a multi-objective optimization problem. A multi-objective optimization problem involves a simultaneous optimization of more than one objective function. The multi-objective function can be stated mathematically as follows:

$$\text{Minimize } v(h) = \begin{bmatrix} v_1(h) \\ v_2(h) \\ \vdots \\ v_\sigma(h) \end{bmatrix} \tag{2.1}$$

where $v_i(h)$, $i = 1, ..., \sigma$ are the σ objective functions in the multi-objective problem. It is unlikely that the different objectives can be optimized by the same parameters. Therefore some kind off trade-off between the criteria in the objective function is needed to ensure a satisfactory results. The concept of "optimality" does not apply directly for multi-objective optimization problems. A useful replacement is the notion of Pareto optimality. Essentially, a vector $\mathbf{h}^*$ is said to be Pareto optimal for the multi-objective function 2.1 if all other vectors $\mathbf{h}$ have a higher value for at least one of the objective functions $(v_i(\cdot))$.

The multi-objective problems are usually solved by combining the multiple objectives into one scalar objective. The solution to the scalar objective is a Pareto optimal point for the original multi-objective function. A standard technique for combining the multiple objectives in a multi-objective problem is to minimize a positively weighted sum of the objectives, that is:

$$\sum_{i=1}^{\sigma} w_i v_i(h), \qquad w_i > 0, \quad i = 1, 2, ..., \sigma \tag{2.2}$$

The selection of the values of the weights w_i is based on the importance of the different objectives and the importance of each objective evaluated by the user.

It is possible to handle the multi-objective function in other ways, e.g. by a multilevel programming. In the multilevel programming the objectives are ordered by importance. Next a set of points that optimizes the most important objective is found. Then the points in this set that optimize the second most important objective are found and so forth until all objectives have been optimized on successively smaller sets.

2.4 Static vs. dynamic dial-a-ride problem

Dial-a-ride transportation systems can be operated according to one of two modes, either static or dynamic.

In static mode all the transportation requests are known in advance. It is therefore possible to plan the actual routes of the vehicles in advance. The static problem can also be used in the long-term decision, strategy, and planning process. In this case the information needed to create the static dial-a-ride problem, is constructed using historical data or forecasted data. Different scenarios are created and solved. In a way the solutions are used in a simulation process. The results are used as a reference giving a better overview of the effects of potential events or trends and the effects of different solution methods.

In dynamic mode the transportation requests are not known, or only partially known in advance. The requests are instead gathered during the planning horizon when the customers call in with their transportation demands. In the dynamic case the actual routes of the vehicles are constructed in real-time. When solving the dynamic problem, the static problem is often solved first, based on requests known beforehand. That solution is then used as a initial solution. This initial solution will then be modified when a new transportation request is received. This modification can be performed by solving the static case over and over again. However it is usually not very efficient so it will be better to use a faster reopimization algorithm to resolve the problem each time a new request is received [11].

The solution methods used in the dynamic case must be very fast since the customer has to be informed about whether his request can be met or not, and if so at what time he is to expect his ride, while he is still on the phone. In the static case however the time used to solve the problem is not nearly as important, since requests are received the day before.

2.5 Inbound vs. outbound customers

The DARP customers can be divided into two groups, namely groups consisting of inbound and outbound customers.

Inbound customers are customers that are located somewhere, perhaps at work, school or hospital, and need to be driven somewhere else, usually home. They need to be picked up after work, school or hospital appointment at a specific point in time. It is also acceptable to collect them a little later than that time but not earlier since they are not ready to leave earlier. There is not a specific time window on their arrival time to their destination location.

Outbound customers are customers that are to be picked up somewhere and then be delivered somewhere else before a certain point in time. For example a person who needs to be driven from her home and to the doctors office, where she needs to be at 3 o'clock. Here the customer can be picked up at any time but the delivery has to be no later than 3 o'clock. Here no specific time windows are associated with the pickup location.

There is therefore either a time window constraint on the pickup or drop off time for each customer in the dial-a-ride system. Usually time windows for both acceptable pickup and delivery times for each customer are defined. The time window, which is not specified by the customer, is derived from the allowed ride time of the customer specified by the transportation operator. A discussion of how the time windows can be set in DARP is presented in section 2.10.4.

2.6 Problem formulation

There are a number of ways to formulate a dial-a-ride problem, which usually depend on the underlying real-life problem. In this project there is no explicit real-life problem to solve and the formulation of the problem can therefore be chosen freely. The problem formulation will however be focused on practical considerations as they are in the Danish transportation sector, see Jørgensen [11].

The problem that will used in this report is formulated in the following way:

- **The dial-a-ride transportation system**
  In this report focus will be set on a DAR transportation system that have customers to be transported from door to door but not necessarily directly. That is, customers are allowed to share a ride but there are no fixed routes. This is for example usually the case in the transportation of elderly and disabled people. All the vehicles start and end their routes at a depot.

- **Static**
  It is usually easier to use a static version of a problem when trying out new solution

methods to solve a problem. That is because the generation of trial data sets is simpler and execution time will be shorter than for dynamic problems. The static version of the dial-a-ride problem is also often used as a foundation for the solution of the dynamic problem. For these reasons it was decided to solve the static version of the dial-a-ride problem in this project.

- **Cost**
  The cost in the DARP is calculated by a multi-objective function. The multi-objective function will be handled by combining the multiple objectives into one scalar objective by minimizing the positively weighted sum of the objectives. The cost of transportation of the customers is estimated in this project to be transportation cost and "cost of bad service".

  Transportation cost consists of transportation time, the total routing time of all the vehicles used in the transportation and the number of vehicles used. The reason for these choices is that it is desirable for the transportation operator to have influence on the length of the routes both in distance and time. The operator wants the vehicles to drive as short as possible since distance has direct influence on the vehicle cost, e.g. gas usage and maintenance. The transportation operator also wants to have influence on the route duration, even though they do not violate the route duration constraint. One could imagine that the transportation operator wants the route duration to be as short as possible, thus being able to hire part-time drivers. If the route duration is shorter, the total routing schedule becomes more robust, meaning that if a driver calls in ill or a bus breaks down the possibility of adding the customers to another route is greater. The operator wants also to be able to see what is the minimum number of vehicles needed to service all the customers. In this project it is though assumed that the number of available vehicles is constant and this segment of the objective function will later be dropped. It is presented here for generalization purpose.

  The cost of bad service is set to be the excess ride time of customers and waiting time of the bus with customers present in the bus. It is decided to use the excess ride time for customers instead of total ride time as a part of an estimate for bad customer service. Excess ride time is the extra time a customer is in the vehicle compared to direct transportation from pickup to drop off locations. That is, the direct ride time is subtracted from the total ride time. The excess ride time gives a better estimate of the customer inconvenience than the total transportation time, since it can be assumed that the customers know approximately how long their direct transportation time is and the customers will not be unhappy at least until that time is exceeded. Another reason for using excess ride time instead of total transportation time is that the direct transportation time is a constant that cannot be decreased. Therefore customers with long transportations have a higher weight than customers with short transportations, even though their inconvenience is not larger than that of others. The reason having the waiting time in an idle bus a part of the bad service is because the longer the customers sit in the vehicle both the cost and unhappiness of customers rice, especially if the bus is waiting idle.

- **Fixed number of vehicles available and no customers rejected**
  The number of vehicles available also has to be decided. That can be performed in two ways. Firstly there can simply be an upper limit on the number of vehicles available and secondly there are no limits on the number of vehicles, but the number of vehicles depends on the needs of the customers. There is a vital difference concerning the customers in this decision. If there is an upper limit on the number of vehicles it is not guaranteed that all customers can be serviced. That raises another question, which needs to be answered, whether all customers must be serviced or if they can be rejected. Those customers, who are rejected, are then to be sent, for instance, by taxi instead. In this project we will simply set an upper limit on the number of vehicles available but customers still cannot be rejected. Rather the number of vehicles is set high enough so that it is not necessary to reject customers. It is considered important to service all the customers using the available vehicles and rather than rejecting customers, time constraints are allowed to be broken.

- **Capacity of the fleet**
  The capacity of the vehicles is also decided beforehand and to keep things simple it is decided to have a constant capacity for each vehicle available equal to the number of seats in the vehicle.

- **Special needs at each stop**
  In this project there is one fixed capacity for each vehicle so a special seat equal a specific number of regular seats. At each stop there is a demand in number of seats. In this way it is also possible for customers to have extra passengers travelling with them. There is also service times that correspond to each stop, i.e. customer, which gives the possibility of assisting the customer getting in and out of the bus.

- **Maximum ride time of customers**
  An upper limit on the time the customer is allowed to sit in the vehicle is defined.

- **Time windows for each stop**
  A time window for all stops, which can be specified either by the customer or the transportation operator, is defined.

- **Maximum route duration**
  A maximum on the length of the route duration, i.e. the time it takes the vehicle to leave the depot, service all the customers on its route and return to the depot again is set. This maximum route length can for example correspond to the shift length of the drivers - as is the case in this project.

Constraints that will not be included into the model are for example constraints concerning union rules and even placements of customers on the routes. Placing the customers evenly is desirable since it evens out the workload on the drivers. Costs that will not be included are fixed costs, such as capital cost, fixed costs for vehicles and depots, salary costs (assumed constant number of staff), etc. The shape of the route will not be included into the dissatisfaction measurement of the customers.

2.7 Notation in the Mathematical Model

First lets assume that we have a set of n customer requests. Each request specifies a pickup location, i, and delivery location, $n+i$. The customers also specify their demand, dem, which is the number of seats required for the passengers that are to be transported from location i to $n+i$ at the same time, and either a preferred pickup time, a_i, or drop off time, b_{n+i}. Each transportation vehicle, k, starts in an origin depot $o(k)$ and ends in a destination depot $d(k)$ and each vehicle has a constant capacity C^k.

Now we can define the following sets:

$P = \{1, \ldots, n\}$	set of pickup locations
$D = \{n+1, \ldots, 2n\}$	set of delivery locations
$N = P \cup D$	set of pickup and delivery locations
K	set of vehicles
$V \subset K$	set of vehicles used in solution
$A = N \cup \{o(k), d(k)\}$	set of all possible stopping locations for all vehicles $k \in K$

We also define the following parameters:

a_i	earliest time that service is allowed to start at in location i
b_i	latest time that service is allowed to start at in location i
s_i	service time needed at location i
$t_{i,j}$	travelling time or distance from location i to j
l_i	change in load at location i
r^k	maximum route duration for vehicle k
u_i	maximum ride time for customer with pickup location i

The following decision variables will be used in the model:

$$x^k_{i,j} = \begin{cases} 1, & \text{if vehicle } k \text{ services customer at location } i \text{ and next customer at location } j \\ 0, & \text{otherwise} \end{cases}$$

m	number of vehicles used in solution, i.e. $\lvert V \rvert = m$
T^k_i	time at which vehicle k starts its service at location i
L^k_i	load of vehicle k after servicing location i
W^k_i	waiting time of vehicle k before servicing location i

In the model the weights in the objective function will be the following:

w_1	weight on customers transportation time
w_2	weight on number of vehicles used
w_3	weight on route duration
w_4	weight on customers excess ride time
w_5	weight on waiting time for customers

2.8 Mathematical model

The resulting mathematical model then becomes:

Minimize

$$\begin{aligned} & w_1 \sum_{k\in V}\sum_{i,j\in A} t_{i,j}x^k_{i,j} + w_2 m + w_3 \sum_{k\in V}(T^k_{d(k)} - T^k_{o(k)}) + \\ & w_4 \sum_{k\in V}\sum_{i\in P}(T^k_{n+i} - s_i - T^k_i - t_{i,n+i}) + w_5 \sum_{k\in V}\sum_{i\in N} W^k_i(L^k_i - l_i) \end{aligned} \tag{2.3}$$

Subject to

$$\sum_{k\in V}\sum_{j\in P\cup d(k)} x^k_{o(k),j} = m \tag{2.4}$$
$$\sum_{k\in V}\sum_{i\in D\cup o(k)} x^k_{i,d(k)} = m \tag{2.5}$$
$$\sum_{j\in A} x^k_{i,j} - \sum_{j\in A} x^k_{j,i} = 0 \qquad \forall k\in V, i\in N \tag{2.6}$$
$$\sum_{k\in V}\sum_{j\in N} x^k_{i,j} = 1 \qquad \forall i\in P \tag{2.7}$$
$$\sum_{j\in N} x^k_{i,j} - \sum_{j\in N} x^k_{j,n+i} = 0 \qquad \forall k\in V, i\in P \tag{2.8}$$
$$x^k_{i,j}(T^k_i + s_i + t_{i,j} + W^k_j - T^k_j) \le 0 \qquad \forall k\in V, i,j\in A \tag{2.9}$$
$$T^k_i + s_i + t_{i,n+i} + W^k_j - T^k_{i+n} \le 0 \qquad \forall k\in V, i\in P \tag{2.10}$$
$$a_i \le T^k_i \le b_i \qquad \forall k\in V, i\in A \tag{2.11}$$
$$x^k_{i,j}(L^k_i + l_j - L^k_j) = 0 \qquad \forall k\in V, i,j\in A \tag{2.12}$$
$$l_i \le L^k_i \le C^k \qquad \forall k\in V, i\in P \tag{2.13}$$
$$L^k_{o(k)} = L^k_{d(k)} = 0 \qquad \forall k\in V \tag{2.14}$$
$$T^k_{d(k)} - T^k_{o(k)} \le r^k \qquad \forall k\in V \tag{2.15}$$
$$T^k_{n+i} + T^k_i \le u_i \qquad \forall k\in V, i\in P \tag{2.16}$$
$$x^k_{i,j} \in \{0,1\} \qquad \forall k\in K, i,j\in A \tag{2.17}$$

2.9 Objective function

The objective function 2.3 of the dial-a-ride problem is a multi-criteria objective function. The objective function consists of the competing objectives of minimizing the total transportation cost, i.e.

$$\sum_{k\in V}\sum_{i,j\in A} t_{i,j}x^k_{i,j} \quad \text{and} \quad m \quad \text{and} \quad \sum_{k\in V} T^k_{d(k)} - T^k_{o(k)}$$

and the inconvenience of customers, i.e.

$$\sum_{k\in V}\sum_{i\in P} T^k_{n+i} - s_i - T^k_i - t_{i,n+i} \quad \text{and} \quad \sum_{k\in V}\sum_{i\in N} W^k_i(L^k_i - l_i)$$

The total transportation cost is here estimated to be proportional to the total time used when transporting the customers by all the vehicles, the total number of vehicles used in the solution and the total route time of all vehicles used. The customer inconvenience is estimated to be proportional to the total excess ride time for the customers and the total waiting time for the customers in the vehicles.

In order to handle this multi-criteria objective function each part of the objective function is multiplied by a variable and added. These variables are called w_1, w_2, w_3, w_4 and w_5. The values of the variables are then used to decide the weight of each criteria in the overall problem.

2.10 Constraints

The constraints in the model can be divided into five groups: Depot, routing, precedence, timing, and capacity constraints.

2.10.1 Depot constraints

The depot constraints describe the requirement that each vehicle starts and ends in a depot. The depot constraints are represented by constraints 2.4 and 2.5 in the mathematical model. That is, the number of vehicles that exit the origin depots and enter the pickup locations and destination depots is the same as the number of vehicles that enter the destination depots from the drop off locations and origin depots. In these constraints a vehicle is allowed to leave a origin depot and drive straight to a destination depot. The reason for this is that it gives the possibility of not using an available vehicle, it stays in the same depot. The possibility of reallocating a vehicle from one depot to another is also open and then the vehicle has a new origin depot in the next planing horizon.

2.10.2 Routing constraints

The routing constraints 2.6 simply require that all locations are to be visited. They ensure that there are equally many vehicles that drive to a location as drive from the same location. There is also an upper limit on the route duration of each vehicle. This constraint is represented by equation 2.15 in the model.

2.10.3 Precedence constraints

The precedence constraints represent the requirement that each customer must first be picked up at his pickup location and then dropped off at his delivery location by the *same* vehicle.

This is handled by the constraints 2.7 and 2.8. The first of these constraints makes sure that there is exactly one vehicle which leaves every origin location, i.e. every request is

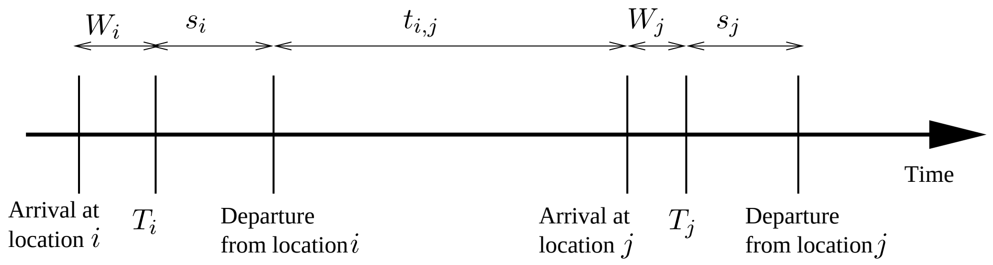

Figure 2.1: The time axis used in the model. A vehicle arrives at location i and has to wait for time W_i. The servicing in location i starts at T_i and takes time s_i. The vehicle departures location i at time $T_i + s_i$ and arrives at the next location j at time $T_i + s_i + t_{i,j}$, where $t_{i,j}$ is the direct transportation time from i to j.

met. The second set of constraints state that the origin and destination locations of a customer are in the same trip. When these two constraints are considered together along with the routing constraint (equation 2.6) it can be seen that they make sure that each customers pickup and drop off locations are visited once and only once by the same vehicle.

In order to obtain a feasible solution a compatibility constraint is introduced. Constraint 2.9 make sure that the arrival time at at location j $(T_j^k - W_j^k)$ must be larger than the sum of departure time from location i $(T_i^k + s_i)$ and travelling time, $t_{i,j}$, between the locations if that leg is to be part of the route. Figure 2.1 shows the time axes used in this model.

Additionally to have a feasible solution it is necessary to visit first the origin point of a customer and then the delivery point. That is, the arrival time at $n + i$ must be larger than or equal to the sum of the departure time from location i and the travelling time, $t_{i,n+i}$, between the locations. This results in the constraints 2.10.

2.10.4 Timing constraints

In this model time windows for all pickup and delivery locations are defined. It means that the transporting vehicle has to enter the location within a specified time period and start servicing the customer. The time windows are introduced into the model through the constraints 2.11.

Definition of time windows

The time windows are defined to be the interval $[a_i, b_i]$ for the pickup locations and $[a_{n+i}, b_{n+i}]$ for the drop off locations $(\forall i \in P)$. The vehicle has to start servicing the customer within these time intervals. That means that it is legal for a vehicle to departure a location at later time than the upper time window states. This is the case if the time difference is used servicing the customer entering or leaving the bus at that location.

The time windows for inbound customers (see definition in section 2.5) are set according to the customers requests on earliest pickup time. These desired times then set the lower limit for the pickup time window, i.e. equal to a_i. Usually the transportation operator or the authority providing the service specifies a time limit on the maximum deviation from these desired times, dev, often 10-30 minutes. Then the upper limit on the pickup time window is set as: $b_i = a_i + dev$.

The lower limit on the delivery time window is then the earliest possible arrival time, i.e. the time at which the customer could arrive to the destination if the customer is picked up at the earliest pickup time, serviced and transported directly from the origin to destination. That is: $a_{n+i} = a_i + s_i + t_{i,n+i}$.

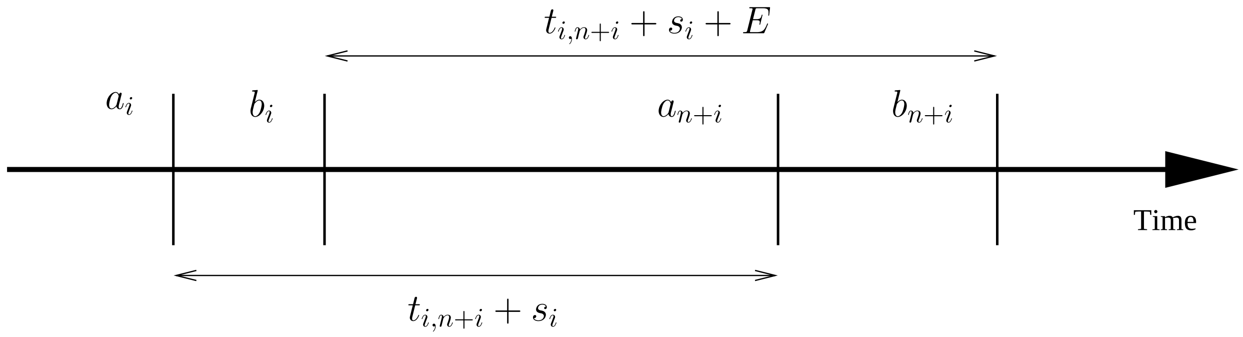

Figure 2.2: Setting the time windows for the drop off location $[a_{n+i}, b_{n+i}]$ given the pickup locations time window $[a_i, b_i]$, the direct transportation time $t_{i,n+i}$ from i to $n+i$, service time s_i at i and the upper limit on excess ride time E.

The upper limit is set as the latest possible feasible arrival time for the customer, for which the constraints on ride time and pickup time are fulfilled. The ride time constraint can be formulated using the concept of maximum excess ride time. The excess ride time is the difference in actual ride time and direct transportation time. Usually an upper limit on the excess ride time, E, is specified for the customers. Excess ride time is the extra time the customer has to sit in the vehicle compared to being transported directly from pickup location to drop off location. This excess time is often specified as a linear equation, e.g. 5 minutes + $\frac{1}{2}$ direct transport time. In this model there is a constraint on the total time each customer is allowed to sit in the vehicle, see constraint 2.16. This allows to set different criteria for different customers. Then the upper time limit for the drop off location becomes: $b_{n+i} = b_i + s_i + t_{i,n+i} + E$. These time window calculations are shown in figure 2.2.

In the case of an outbound customer, the customer specifies the latest drop off time, which is set as the upper limit on the drop off time window, equal to b_{n+i}. Then the other time window limits are found using the same method backwards in time.

2.10.5 Capacity constraints

The vehicles used to in transportation of the passengers have a fixed capacity, C^k, representing the number of passengers (in seats) that can be transported by that vehicle at the same time. These capacity constraints cannot be exceeded at any time and are represented by constraint 2.13 in the mathematical model.

Loads on vehicles

In order to keep track of the number seats needed for the passengers in each vehicle throughout the route the term of load for each vehicle is introduced. Load of vehicle at a point in time is the number of seats needed for the passengers in the vehicle at that point in time. When a vehicle has serviced a pickup location i the change in load is represented by $l_i = dem_i$ and the change in load after servicing a drop off location $n+i$ is $l_{n+i} = -dem_i$.

The actual load of a vehicle k after servicing a location i is L_i^k. The load of the vehicle after servicing the next location in the route, j, is then $L_j^k = L_i^k + l_j$. In order to calculate the load in this manner constraint 2.12 is introduced to the model. It ensures that the loads are correctly calculated for the edges used in the route. The actual loads of the vehicles are set to zero at the depots in constraint 2.14.

2.10.6 Linearization of constraints

Note that constraints 2.9 and 2.12 are non-linear constraints. They can be linearized, which is necessary if an LP-solver is to be used to solve the problem. The linearization of constraint 2.9 can be performed in the following way:

$$x_{i,j}^k(T_i^k + s_i + t_{i,j} + W_j^k - T_j^k) \leq 0 \tag{2.18}$$

is equivalent to

$$T_i^k + s_i + t_{i,j} + W_j^k - T_j^k - \mathbb{M}(1 - x_{i,j}^k) \leq 0 \tag{2.19}$$

where $\mathbb{M}$ is a large number and recall that $x_{i,j}^k$ is binary.

Constraint 2.12 can also be linearized. First the constraint needs to be replaced by two inequality constraints, which combined are equivalent to the equality constraint 2.12. The two inequalities are:

$$x_{i,j}^k(L_i^k + l_j - L_j^k) \leq 0 \tag{2.20}$$

$$x_{i,j}^k(L_i^k + l_j - L_j^k) \geq 0 \tag{2.21}$$

The linearization of the two inequalities 2.20 and 2.21 is performed in the same manner as is shown above for constraint 2.9.

2.11 Related problems

It is very useful to study the problems that are related to the DARP and the solution methods used to solve the related problems. Solution methods that give good results for a related problem can be expected to give good results for the DARP as well.

The travelling salesman problem (TSP) gets its name from a salesman who has to drive from door to door to visit his predefined customers trying to sell his merchandise. He leaves his house, visits all the customers and returns back to his home after the work is performed. He, of course, wants to return home as early as possible. The length of the workday of the salesman is determined by the order of the visits to the customers. In this statement we are taking two assumptions. The first assumption is that the travelling time between any two customers is not time dependent, i.e. the travelling time is the same in the morning, afternoon or night. The second assumption states that the travelling time between any two customers is independent on the order of the customers, i.e. travelling time from customer i to customer j is the same as the travelling time from customer j to customer i. The TSP is then to find the route in which all the customers are visited with the minimum total travel time.

Of course the TSP can be generalized to fit other systems in which it is necessary to get from a base location, visit other locations once and only once and end at the base again. The problem stated more generally is then to find a route for a vehicle which visits each of n predefined locations once and only once and minimizes the total travel time. The travel time is dependent on the speed of the salesmans vehicle and the speed can vary, e.g. on different types of roads, and therefore it is often the travelling distance that is minimized instead of travel time in TSP. It seems easy to solve such a problem but if there are n locations to visit there are $(n-1)!$ possible solutions to the problem.

This is the classical travelling salesman problem. Additional constraints and specifications can be added to the problem, as in **the vehicle routing problem (VRP)**. In VRP more than one vehicle is available to visit the locations and each of the vehicles has a limited identical capacity. The home of the vehicles is called a depot, from where the vehicles start and end their route. There can be more than one depot and each having a certain number of vehicles available to visit the locations. Each vehicle is either making deliveries to customers or picking up goods from vendors/plants - not both. Each customer is to be serviced by exactly one vehicle, it is for example not possible for a vehicle to deliver the customer half of the ordered goods and then another vehicle to bring the rest of the order. In this problem the vehicles have a capacity and the distance between customers is known. There can be an upper limit on the length of routes of the vehicles and the demand/supply at each location is known. The vehicle routing problem consists of allocating customers to vehicles and find the order in which each vehicle visits its customer so the total travelling distance of all vehicles serving all customers is minimized while maintaining all the constraints.

In **the vehicle routing problem with backhauls (VRPB)**, which is an extension of VRP, the set of customers is partitioned into two subsets: Linehaul and Backhaul customers. Each Linehaul customer requires the delivery of a given quantity of product from the depot, whereas a given quantity of product must be picked up from each Backhaul customer and transported to the depot. First all the deliveries in the same route must be made and then the pickups in the same route take place, so as to avoid rearranging the goods in the vehicle.

The pickup and delivery problem (PDP), also called the vehicle routing problem with pickup and delivery (VRPPD), is a VRP with the addition that there is a pickup and a delivery location given for each transportation request. So that a vehicle has to drive to a pickup location to pickup goods and then later in the route drive to the corresponding drop off location to deliver the goods that are stored in the vehicle. In other respects there is no other modifications from the VRP described above.

All the problems described above deal with the transport of goods, so issues as how long the goods are in the vehicle are off no or insignificant importance and we can disregard it. This is however not the case with **the dial-a-ride problem**, since it usually deals with the transportation of people. That fact complicates things considerably, since service provided to customers now has to be taken into account. Apart from that the dial-ride-problem is the same problem as PDP described above.

To all the problems described above time window constraints associated with each/some of the locations to be visited can be added. Time windows constitutes a time interval in which one is allowed to visit a specific location. For example, if the time window for customer 3 is $[8, 9]$ it means that the transporting vehicle can deliver/pickup goods between 8 and 9 o'clock. The time windows can be hard constraints, that is if the vehicle does not arrive within the specific time interval it either has to wait, if early, or if late, then the vehicle is not allowed to stop there. Soft time windows allow for visits outside the specific time window but at an extra cost.

The last problem described here is **the bin-packing problem**. The bin-packing problem is the problem of packing a set of objects into a number of bins[2] such that the total weight or volume does not exceed some maximum value. The objective is to arrange the items in such a way that the number of bins used is minimized.

2.12 The Dial-a-ride problem is $\mathcal{NP}$-hard

The dial-a-ride problem can be proven to be $\mathcal{NP}$-hard, see for example Baugh et al. [2]. The proof is based on the related $\mathcal{NP}$-hard travelling salesman problem with time windows, which can be transformed to the dial-a-ride problem. It is assumed that we have given a weighted input graph, G, for TSPTW and by applying the two following rules G can be transformed to a graph G' that is an input graph to DARP:

1. For every node in G add a pair of nodes in G' with the same time windows as the node in G. The pair of nodes representing the origin and destination nodes.
2. For every arc in G add an arc of the same weight to G'. Also add arcs connecting each origin/destination pair of nodes in G' with zero weights.

G' can be constructed from G in polynomial time. If G' has a DARP solution then G has a TSPTW solution and conversely if G has a TSPTW solution then G' has a DARP solution. Since TSPTW is $\mathcal{NP}$-hard DARP is $\mathcal{NP}$-hard.

[2]A bin is a container or an enclosed space for storage.

Chapter 3

Previous Work

Study of the dial-a-ride problem started in the late 1960s. Since then several versions of the dial-a-ride problem have been proposed and several techniques used to solve the problem.

In this chapter a short description of some of the papers that have been published on using heuristics for solving the static multi-vehicle dial-a-ride problem with time windows is presented. Firstly there will be a description of a paper in which the problem is solved using a heuristic algorithm based on insertion. Secondly an introduction of two papers that use meta-heuristics; simulated annealing, and tabu search, to solve the problem is given. A discussion of using the genetic algorithm for solving the related problem of pickup and delivery with time windows and the vehicle routing problem is presented. The chapter will be concluded by a comparison of the papers along with a discussion of their influence on this project.

3.1 Jaw et al.

The work performed by Jaw et al. [9] in 1986 is a pioneer research within the area of dial-a-ride and most of the following research performed in this area is based on their work. In the paper a sequential insertion heuristic algorithm for solving the static dial-a-ride problem is described. The algorithm is referred to as ADARTW.

In the algorithm customers are first sorted according to their earliest pickup time. Then the algorithm tries to assign the customer on the top of the list to a vehicle, that is the customer found to have the earliest pickup time. The assignment is performed by considering the additional cost of all feasible insertions of the customer to a vehicle and the cheapest insertion is chosen. If the customer cannot be assigned to existing vehicles the customer is either rejected or a new vehicle initialized. The algorithm continues to process customers in sequential order according to the list, until the last customer on the list has been processed.

Each customer either has to give a desired pickup time or a desired delivery time. Time windows are then calculated given:

- the desired times,
- the direct ride time from origin to destination for each customer,
- the maximum acceptable ride time for each customer, which is a linear function of the direct ride time,
- the maximum acceptable deviation from the desired pickup or delivery time for each customer.

The objective function combines the minimization of operator costs and the minimization of customer inconvenience with respect to both customer ride time and deviation from desired pickup or delivery times specified by each customer. The different parts of the objective function are balanced by multiplying them by user-specified constants. The values of the constants are then varied in the computational experiments.

In the computational experiments the ADARTW algorithm was run on a number of simulated data sets with 250 customers and 4 to 5 vehicles and real data sets with 2617 customers and 28 vehicles. The CPU time for the simulated data set was about 20 seconds and about 12 minutes for the real data set. The authors conclude that their computational experience shows that the ADARTW algorithm gives at least as good as or superior solutions to those encountered in manual planning in all respects.

3.2 Baugh et al.

In 1998 Baugh et al. [2] presented a paper in which the meta-heuristic algorithm, simulated annealing, is used for solving the static dial-a-ride problem. The authors argue that it is wise to use simulated annealing for solving the problem since it is easily adapted to problems with a well defined neighbourhood structure, it has desirable theoretical convergence properties and it can easily be combined with other meta-heuristics, such as tabu search.

The work is based on the classical cluster-first, route-second approach. A cluster is a group of customers assigned to the same vehicle and also serviced together. Customers are first organized into clusters and then the routes are developed for each individual cluster. In this paper the clustering is performed using simulated annealing while the routing is performed using a modified space-time nearest neighbour heuristic. The authors claim that the clustering is the most vital part of the process since routing only involves a small set of customers and therefore it is not necessary to solve the routing problem with an algorithm as sophisticated as one used for clustering.

The clustering algorithm is initialized by assigning customers to clusters randomly. Two operations are then used to alter the current clustering of customers. Either two customers are randomly chosen and their current clusters are *exchanged*, leaving the total number of clusters unchanged. Alternately, a customer and a cluster are randomly selected, and the customer is *swapped* to the selected cluster, as a result the number of clusters can

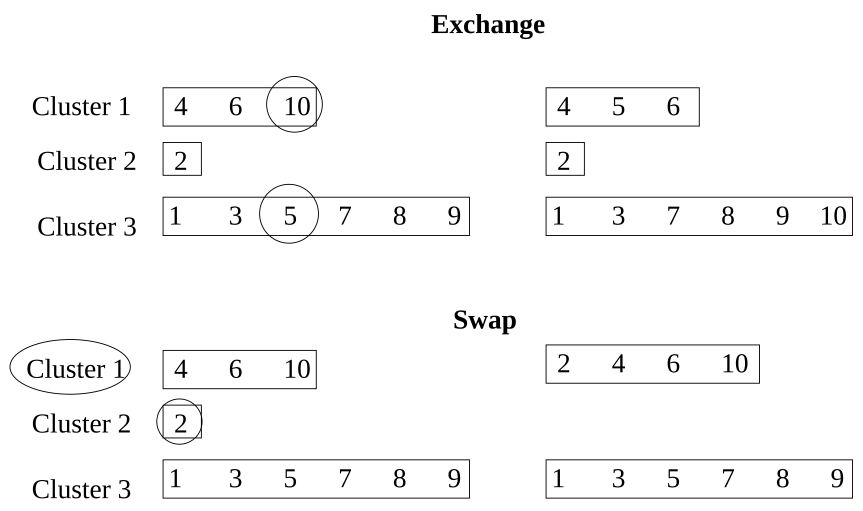

Figure 3.1: An example of the exchange and swap operations used in Baughs clustering strategy. The circles indicate a choice. In exchange two customers are chosen randomly and their cluster assignment exchanged. In swap a customer and a cluster are chosen randomly and the customer is swapped to the chosen cluster.

increase or decrease in the swap. The resulting neighbourhood structure is simple, and it is possible to generate any cluster from any other cluster. The exchange operation results in a smoother solution space, while the swap operation allows for the dynamic change in number of clusters. Figure 3.1 gives an example of the exchange and swap operations for ten customers originally divided into three clusters.

At each iteration of the simulated annealing algorithm, the routing algorithm is invoked on the selected clusters, routes are developed and the new objective value calculated. The objective function is evaluated by the total distance travelled by all vehicles, number of vehicles, and total disutility caused to customers. If the new solution improves the objective it is accepted, otherwise it is accepted with a certain probability. This probability is a function of the change in objective value and the annealing temperature. In order to improve the results of simulated annealing a tabu list is introduced so that accepted transitions are not immediately reversed.

The modified space-time nearest neighbour heuristic used to create routes for each cluster is a greedy algorithm. It starts by visiting the pickup location of the customer with the earliest pickup time. The succeeding location to visit is the location with the lowest cost. The cost is estimated to be the cost of the next three succeeding moves if the location under consideration is visited. The succeeding moves are identified using the space-time separation between the locations that have not been visited and selects the shortest move. The space-time separation is quantified by a weighted sum of the travel time between the locations and the time window violation at the destination location.

The results obtained are based on a set of real-life data set with 300 customers as well as on a generated data set having 25 customers. No CPU times are given in the paper. The authors claim that the algorithm gives near globally optimal solutions. They also state that the method is robust, i.e. it obtains constantly near optimal solutions when run on the test problems. Furthermore it is noted that a minimal user input for fine tuning annealing parameters are needed.

3.3 Cordeau and Laporte

In 2003 Cordeau and Laporte [4] wrote a paper which describes how a tabu search heuristic is used for solving the static dial-a-ride problem. Their algorithm starts with an initial solution that is randomly generated. In each iteration the best solution in the neighbourhood of the current solution is chosen. In order to avoid cycling, solutions possessing some attributes of recently visited solutions, are put on the tabu list and are therefore forbidden for a number of iterations unless they constitute a new incumbent. It is allowed to explore infeasible solutions during the iterations. That is performed by relaxing the constraints in the problem, by adding new terms into the objective function, each of which represents an evaluation of the violation of one constraint multiplied by a positive parameter. After each iteration the parameters are adjusted so that currently violated constraints get more weight in the objective function and the weights are reduced on constraints that are fulfilled by the current solution. The objective function consists of the total transportation cost of the vehicles and the violation terms (customer inconvenience is a part of the violation terms).

The initial solution is constructed by randomly assigning the customers requests to vehicles, and the order of the requests in each vehicles route is the same as the order in which they were assigned to the vehicle. The origin of each request comes first and then the destination.

The neighbourhood of the current solution is constructed using a simple operator that reassigns a request to a new vehicle in the current solution. Now the difference between the new solution and the old solution is restricted to two routes. The order of requests in the two routes is the same as before, but in one of the routes one request is missing while in the other one the same request has been inserted in the route in such a way, that the total increase in total cost for this solution is minimized. The cost for a solution equals the objective function value for the solution. Routes are optimized every time a new best solution is identified, and also systematically during the iteration process. This is performed by intra route exchanges. In the intra route exchanges every customer is removed from its current route and the pickup and drop off locations are reinserted into the route in the best possible positions. The best possible position is the position that minimize the objective function value.

In this algorithm, it is possible to use the full algorithm to evaluate candidate solutions in the neighbourhood. The full algorithm consists of eight steps and in those steps time window violations, route duration and ride times are minimized. It is also possible to take the first six steps, which minimizes time window violations and route duration, and to perform the first two steps and only minimize the time window violations. The three different approaches were tested using both randomly generated data sets with 24 to 144 customers and six real-life data sets containing either 200 or 295 customers. The CPU times for the randomly generated data sets are about 2 minutes for the smallest data sets and up to 93 minutes for the largest data sets. The CPU times for the real-life data sets are given to be about 13 to 268 minutes. The conclusion is that the full version of the algorithm reviled the best solutions but is the most expensive in CPU-time. The results given in the paper are not compared to results obtained by others.

3.4 Jih et al.

Jih et al. [10] published a paper in 2002 on solving the single vehicle pickup and delivery problem with time windows using a family competition genetic algorithm.

When using the genetic algorithm it is necessary to have a chromosome representation of a route and that is performed by letting the chromosome represent the locations in a travelling sequence of the route.

The algorithm is allowed to explore infeasible solutions during the iterations process. The objective function, in GA-germs also known as the fitness function, is the summation of the total travel cost of the vehicle and the penalty for violating constraints, which is the case with infeasible solutions.

The family competition genetic algorithm is based on the genetic algorithm with the extension that every individual also owns its family. When creating a new generation each individual in the current population is set to be a family father. Each family father is used to create a new family by recombining the family father and randomly chosen alternative parents from the population. The recombining is called a crossover in GA terms. The crossover can be followed by a mutation, which is usually a small random change of a solution. The size of a family is a constant k_f. In each iteration $k_f M$ new individuals are created, where M is the population size. Only the best individual in a family survives and becomes a member of the new population. The iterations will run until a termination condition is reached. In the termination condition, used in the experiments that are presented, is not stated.

Two types of operators are used to change the current solution: Crossover and mutation. Four different types of crossover are considered; the order-based crossover, uniform order-based crossover, merge cross #1, and merge cross #2. The first two are traditional crossover operators but the last two use a global precedence vector to give guidance in the crossover. Two mutation operators are considered. The first one selects two random genes and interchanges their position, while the second one chooses randomly two cut sites and the order of the sub-routes is inverted. Mutation is used in this algorithm if a child and one of its parents represent the same route. The reason for this choice is that it supports route diversity and prevents the search space to become bound in a local optimal solutions.

The algorithm is run on randomly generated data sets with up to 100 customers and the CPU time is about 38 minutes for the largest data sets. In the experiments the family competition genetic algorithm is compared to the traditional genetic algorithm. The authors found that the first one results in better solutions and the probability of obtaining the best solution is higher at similar running times. The different types of crossover operations are also compared and the order-based crossover is found not to be suited for the pickup and delivery problem. It is found that the uniform order-based crossover gives the best solutions but requires much execution time while both types of the merge crossover are faster but give reasonably good solutions and might therefore be better suited for real-time approach. The main conclusion is that the family competition genetic algorithm succeeded in finding feasible solutions to the generated problems in reasonable time and that the choice of modifying operators greatly influences the results of the algorithm. The results of the algorithm are compared with the optimal values which are available for the smaller data sets (up to 40 customers). The best results for the family competition genetic algorithm is by using the uniform order-based crossover is able to reach the optimum on the average in 83% of the runs for the data sets with up to 40 customers.

3.5 Pereira et. al

The paper written by Pereira et. al [13] is called: “GVR: a New Genetic Representation for the Vehicle Routing Problem.” In the paper the genetic algorithm is used to solve the capacitated VRP. A two-level schema (GVR) is designed to represent all the information a potential solution must encode. A potential solution must specify the number of vehicles required, allocation of customers to vehicles and the order of customers in the route of each vehicle.

An individual represents a solution and is made of a chromosome. The genes in the chromosome are the customers in their visiting order. Each customer must be represented exactly once in the individual. If capacity of the vehicles is exceeded in any route, the route is split up into smaller routes, i.e. new vehicles are added to the solution, until capacity is within limits, at the interpretation level.

The algorithm proceeds from an initial population of n individuals. In each iteration there are chosen n parents and n offsprings created using genetic operators. Two types of operators are considered: Crossover and mutation. The operators should be capable of changing the order of customers within routes, modifying the allocation of customers to vehicles and altering the number of routes in solution. The offspring must also represent a legal solution. A legal solution is a solution in which each customer is assigned to exactly one vehicle. The capacity of the vehicles is not an issue since it is assumed that the capacity constraint is checked and fixed at interpretation level.

In the crossover an offspring is created by inserting a fragment of genetic material (a sub-route) from one parent into one line of the chromosome (route) of the other parent. The placement of the insertion is directly behind a customer that is not a part of the sub-route and is closest to the first customer in the sub-route. Afterwards duplicates in other chromosomes are removed. An example of how the crossover works is given in

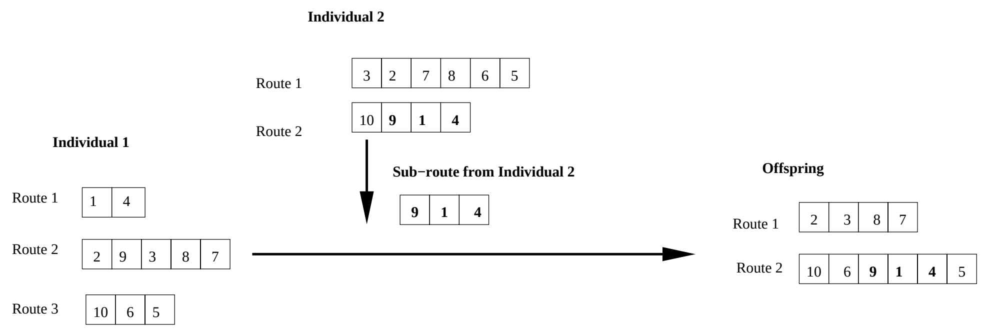

Figure 3.2: An example of the crossover used by Pereira et al. An offspring is created by selecting a sub-route from parent 2, inserting it into parent 1 and removing duplicates. Here it is assumed customer 6 is closest to customer 9.

figure 3.2. In the figure individual 1 and 2 have been chosen as parents. A sub-route is randomly selected from individual 2. The customer that is geographically closest to the first customer (6) in the sub-route and is not a part of the sub-route is identified (9). The sub-route is inserted into individual 1 and the placement is directly behind customer 9. The customers that originally belonged to individual 1 and are now duplicates of the customers in the sub-route are removed from the offspring. The crossover is capable of reducing the number of routes, changing the order of customers in routes and reallocating customers to routes. It is however not possible to add new routes to the solution.

The offsprings can be mutated after the crossover. In this paper there are four mutation operators. First, two customers can be swapped within the same route or different routes. Second, routes can be inverted, i.e. the visiting order of customer is inverted. Third, a customer is selected and inserted in another place, possibly creating a new route containing only this customer. Fourth, a sub-route is chosen and inserted in another random place, both intra or inter-displacement are a possibility. The fourth mutation operator is very similar to the crossover, the only difference is the selection of insertion placement of the sub-route. In the crossover it is behind the geographically closest customer but in the mutation it is chosen randomly. The mutation operator is therefore capable of adding and deleting routes, altering order and allocation of customers.

The algorithm is tested on a collection of data sets with 12 instances from some well-known benchmarks[1]. The results show that the method is very efficient for solving this problem. The authors are able to find reach the best solutions that have been found for most instances of the well-known benchmarks or be very close to the best. They are even able to find new best solutions to some of the test instances. The method proved to be robust, i.e. parameter settings do not affect the quality of the solutions obtained. No CPU times are given in the paper. It is however concluded that the results are to be considered preliminary.

[1]Augerat Set A, Augerat Set B and Christofides and Elion

3.6 Comparison

In this section a comparison of the papers, which have been described previously in this chapter, will be presented. There will also be a discussion of the ideas and inspiration that these papers have had on the work described in this project.

The first paper is written by Jaw et al. in 1986. The paper describes a pioneering work performed in the area of the dial-a-ride problem. A sequential insertion heuristic algorithm to solve the static dial-a-ride problem is proposed. In this algorithm the customers are sorted according to their earliest pickup times. The algorithm is used to solve randomly generated test instances with up to 250 customers in about 20 seconds and real-life data sets with 2617 customers in about 12 minutes. The results of the real-life data sets are compared with results from manual planning and concluded that the solution method gives as good as or superior solutions to the solutions obtained by manual planning.

The paper is, in my opinion, very clearly written and interesting. It explained very well the time windows calculations and the complications involved in the dial-a-ride problem.

The second paper, written by Baugh et al. in 1998, describes how the DARP is solved using simulated annealing to group customers thereafter a space-time nearest neighbour heuristic is used to make routes for each group. To improve the performance of the simulated annealing schema, a tabu list is included. The simulated annealing is used to solve randomly generated test instances with up to 25 customers and real-life data sets with up to 300 customers but no CPU times are given. The results obtained for the real-life data sets are compared to the plans used in practice. The results obtained by Baugh et al. outperformed the plans used in practice. It should however be noted that the plans are not optimal and include several policies that are not modeled by Baugh et al. Therefore I conclude that it is impossible to conclude about the quality of the solutions obtained. The authors do however claim that the algorithm is capable of giving near globally optimal solutions but that is not shown explicitly. The algorithm is further claimed to be robust, i.e. obtains constantly good optimal solutions, and to require minimal user input for fine tuning the parameters needed in the algorithm.

The paper is very well written. The terms used in the paper are clearly and precisely explicated. As an example a mathematical model of the dial-a-ride problem is presented, which is not seen in many papers on the subject. Parts of that model are used in the mathematical model presented in this project. The ideas of clustering and routing will be used in solution method and the routing heuristic presented in the paper lays the foundation for the routing heuristic used in this project. It is however questionable whether it is wise of the authors to use simulated annealing with a tabu list instead of simply using the tabu search heuristic. The fact that the authors do not give CPU times and explain how they come to the conclusion that their algorithm is able to give near optimal solutions decreases the credibility of the paper in my opinion.

The third paper is written by Cordeau and Laporte in 2003. There is a description of how a tabu search heuristic can be used to solve the DARP. The method is tested on several randomly generated data with up to 144 customers in abut 93 minutes sets along with some real-life data sets with up to 295 customers in 268 minutes. The results are not compared to results obtained by others and only the cost is presented as a result for each data set.

The paper is not as well written as the previous ones. It is harder to understand the paper and catch precise information details. Furthermore, in the results chapter there is also only reported the cost for the best solutions found and the results not compared with results from others. Consequently it is impossible to compare the results with results in the previous papers. However an Internet address where all the data is available along with more detailed results is given. It is excellent that the test cases are available for others to use as to being able to compare results and it is perhaps the beginning of a database of standard dial-a-ride problems. All the test cases that will be used in this project are received from this Internet site.

The fourth paper, written by Jih et al. in 2002, gives a description of how a family competing genetic algorithm is used to solve the single vehicle pickup and delivery problem. The algorithm is used to solve randomly generated data sets with up to 100 customers using up to 38 minutes. For the smaller data sets the algorithm is able to find the optimum with a probability of 83% on the average. The conclusion of the authors is that the family competing genetic algorithm succeeds in finding feasible solutions in reasonable time.

The paper is very interesting but very sparse in its description of the methods used to solve the problem. Also only the single vehicle PDP with time windows is solved and it is not clear how to extend the method to the multi vehicle case. The paper gave inspiration and ideas of how to use the GA in solving the dial-a-ride problem at the initial stages of the working process of this project.

The fifth paper is written by Pereira et al. in 2002. Here an analysis of how the genetic algorithm is used to solve the vehicle routing problem is presented. A description of a two level representational schema is given. The schema is constructed in order to incorporate all the information that a candidate solution must encode. Experimental results show the method to be both effective and robust as well as capable of discovering new best solutions to some well-known benchmarks but no CPU times are given.

This paper is very well written. There are many examples and illustrations of how the algorithm works. The two level representational schema and the crossover operators presented in the paper gave an inspiration of how to use the genetic algorithm for solving the DARP presented in this project.

The first three of the papers using different solution methods to solve the static dial-a-ride problem. The formulation of the DARP is not the same in the papers and they use different solution methods. The results obtained in these papers are not compared to optimum or best known solutions. It is therefore impossible to compare the effectiveness of the solution methods introduced in the three papers. The last two papers introduce solution methods to problems related to the DARP and the results they report are therefore not comparable to the results in the first three papers.

The work presented in this project is mainly based on these five papers along with the PhD theses about the dial-a-ride problem, written by Jørgensen [11] in 2002. In the theses the dial-a-ride problem is described in detail which has been great help in getting a deeper understanding of the various aspects of the DARP.

A summary of the ideas and inspiration that the papers have contributed with to this work, is presented in table 3.1.

Table 3.1: Overview of the contributions of the papers to this work

Paper by:	Contributions to this work
Jaw et al.	Understand of the time windows calculations
Baugh et al.	Gave the idea to use cluster-first, route-second to solve the formulated problem and the routing heuristic will be used for routing
Cordeau and Laporte	Test problems used to test the heuristic developed in this project
Jih et al.	Gave the idea to use the genetic algorithm to solve the dial-a-ride problem
Pereira et al.	Gave further ideas of how to use the genetic algorithm to solve the formulated problem

The reasons for choosing the genetic algorithm for a solution method in this project are explained in chapter 4.

Chapter 4

Solution method

In this chapter a description of the solution method used to solve the dial-a-ride problem formulated in chapter 2 is given. The solution method is based on the cluster-first, route-second principle. A genetic algorithm is developed for solving the clustering and the modified space-time nearest neighbour heuristic developed by Baugh et al. [2] is used in an adjusted version for solving the routing.

The chapter starts by a description of the decision process of choosing the solution method. Next the genetic algorithm will be described containing a discussion on the important issues to consider when constructing the genetic algorithm, such as solution representation, population size, construction of initial population, fitness calculations, selection mechanism and which modifying operators to chose. The two types of modifying operators addressed are crossover and mutation. Thereafter the modified space-time nearest neighbour heuristic will be described.

4.1 Choosing the solution method

The choice of solution method for solving the dial-a-ride problem depends on solution techniques considerations (such as existing algorithms in the available literature), available computer software, computer/hardware and time.

In order to limit the search for existing algorithms to solve the problem, it is first considered whether to use exact or heuristic algorithms. Lets start by comparing the two types of algorithms.

When solving an optimization problem we are really only optimizing a model of a problem originating in the real-world. There is no guarantee that the best solution to the model is also the best solution to the underlying real-world problem. Two reasons for why the best solutions do not match are that some real-world constraints can be decided to be omitted in the model, because they are not considered important, and the numbers used in the model/implementation are not be precise to the last digit (e.g. π and measurements). Even though heuristics are not guaranteed to provide us with an optimal solution to the underlying problem then neither are exact methods. Furthermore heuristic methods are usually more flexible and capable of coping with more complicated and realistic objective functions and/or constraints than exact algorithms. Another reason for considering heuristics for solving the dial-a-ride problem is the $\mathcal{NP}$-hardness of the DARP, see section 2.12, which makes the problem solvable using exact algorithms only for small problem sizes.

In Baugh et al. [2] alternative mathematical programming approaches are explored for solving a practical DARP. Three mixed-integer linear programming models (MILP) are tested and it concluded that the largest problem size that can be solved using commercial optimization packages is about 10 customers. The MILP is solved using a custom branch-and-bound solver and it is observed that for problem sizes larger than 15 customers even finding any feasible integer solution proves to be difficult. A dial-a-ride system consisting of 15 customers is small compared to a realistic system of around 200-300 customers.

Next we need to consider which heuristic algorithm to use. As we can see in Cordeau and Laport [4] as well as in Baugh et al. [2], meta-heuristics work well when solving the dial-a-ride problem. According to Jih et al. [10] the genetic algorithm (GA) is a good approach for solving $\mathcal{NP}$-hard problems and in particular the related travelling salesman problem and vehicle routing problem.

One of the main objectives of this project is to experiment with a new solution method for the dial-a-ride problem and the GA has not previously been used for solving the DARP.

Since the GA is considered to be a good approach for solving routing problems and has not been used for solving the DARP it is decided to investigate the behaviour of the genetic algorithm when used for solving the dial-a-ride problem formulated here. The original idea is solving the problem in one step using a sequential two-level chromosomes representation, similar to the one used in Pereira [13]. In the chromosome representation both customers allocation to vehicles and the order of customers in the routes of the vehicles are encoded. The chromosome representation is used for solving the vehicle routing problem but the extension to the dial-a-ride problem is problematic. The main problem is to check if the precedence constraint is kept. In order to solve this problem some elaborate fix-up procedure would have to be initialized each time a new individual is created. That is a complicated and time consuming process so other ideas are considered. The idea of the cluster-first, route-second strategy comes to mind inspired by papers written by Baugh et al. [2] and Toth and Vigo [17]. In the papers the cluster-first, route-second solution method is described, and Baugh et al. applies it to the DARP while Toth and Vigo apply it to the VRPB. The cluster-first, route-second strategy bears some similarities to the bin-packing problem. In the bin-packing problem objects are to be packed into a number of bins, that is the objects are organized into clusters, one for each bin. The way the objects are arranged into the bins can be decided simultaneously or decided after the objects have been clustered. This is the same idea as the cluster-first, route-second in DARP.

The cluster-first, route-second is a strategy that partitions the entire set of customers into clusters. All the customers belonging to the same cluster are serviced by the same vehicle. Routes are thereafter developed for the individual clusters.

Based on the good experience with the cluster-first, route second strategy reported by others [2] on similar problems it is decided to use cluster-first, route-second solution approach. First it is considered using the GA for both clustering and routing, but since the GA behaves in a random manner it seemed like a better idea to use the GA for one part and another more deterministic method for the other part of the solution method. The most crucial decision for DARP is the allocation of customers to vehicles. Therefore it is vital to use a good global solution technique, such as the GA, in that part of the problem while the routing can be solved by a faster heuristic. Very good heuristics algorithms for solving the routing problem to near optimality exist, such as the Lin-Kernighan algorithm [2]. Note however that the routing problems are expected to be small, typically the number of customers in a route is expected to be of the order 10 or 20. The modified space-time nearest neighbour heuristic by Baugh et al. should give good solutions with minimal computational requirements [2].

The problem will be solved using the genetic algorithm for clustering and a modified space-time nearest neighbour heuristic for routing

4.2 The Genetic Algorithm

The main ideas behind the genetic algorithm are taken from biology. Terms such as natural selection and the survival of the fittest lie at the core of the genetic algorithm (GA). In the GA, a population of individuals is created, each individual represents a candidate solution to the problem. Such a candidate solution is represented by a string of numbers, similar to the genetic representation of chromosomes for living organisms. So an individual is represented by its genetic material.

The individuals are paired and offsprings produced, that is, new candidate solutions are created using existing solutions. The selection of individuals to pair and produce offsprings is based on the fitness of the individuals, i.e. the fitter the individual is, the higher probability of the individual to be allowed to produce offsprings. The fitness values of individuals are usually proportional to the corresponding objective function value of the solution the individual represents. The individuals used to produce an offspring are called parents. The new offsprings are created using some kind of genetic operators. The two most common genetic operators are crossover and mutation.

- **Crossover**
  Crossover is the main genetic operator and is applied to pairs of individuals. In crossover, an offspring is created by combining the genetic material of two existing solutions, e.g. the first part of one parent chromosome and the last part of the

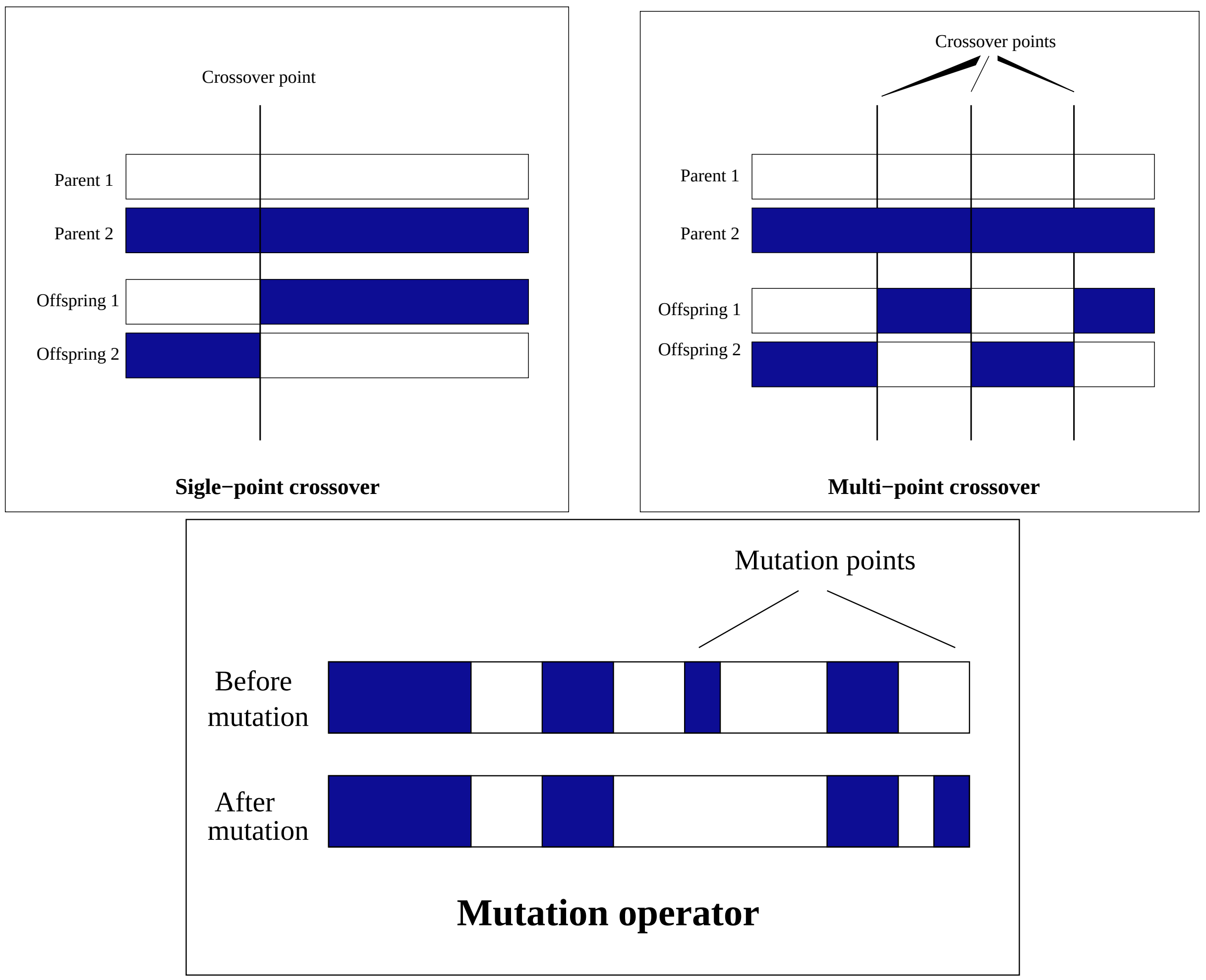

Figure 4.1: A schematic presentation of a simple crossover and mutation. In the single-point crossover a crossover point is chosen and an offspring inherits the section of one parents genes before the crossover point and second parents genes after the crossover point. Multi-point crossover is similar. In the mutation two mutation points are chosen and the genes interchanged.

second parents chromosome. The idea here is that through selective breeding[1], the offsprings will inherit the good qualities of their parents which combined gives an even fitter individual than either parent. If that is however not the case, i.e. the offspring turns out to be inferior to the parents, then the offsprings chances of survival in the next generations are small (as will be explained later). The good offsprings on the other hand will have a higher chance of survival and their characteristics will be spread throughout the entire population. The fitness of the entire population is therefore expected to increase as the iterative process progresses.

- **Mutation**
  Mutation is a genetic operator that is applied to a single individual. An individual is mutated usually after crossover with a small probability. In mutation a modification of the genes of the offspring that is not inherited from either parent is made.

[1]Selective breeding is a process of selecting parents in order to produce offsprings that possess desirable characteristics that will benefit the species in the long term.

Mutation can therefore introduce new characteristics which are not present in any part of the parents population. Mutation is often said to be the raw material of evolution for living organisms and this is what GA tries to capture. The purpose of mutation is to escape from one region of the solution space to a completely new region. If there is no mutation in the process it is hard or impossible to investigate the whole solution space and it is not possible to prove that the GA without mutation will settle in a global or even a local optimum, which will be discussed in greater detail in section 4.2.3.

For a more detailed discussion about specific genetic operators, see section 4.2.11 and for an example of simple crossover and mutation operators, see figure 4.1.

The GA is an iterative procedure. It is started by creating an initial population of individuals and their fitness values calculated. Then individuals are chosen to mate and produce offsprings through crossover. There is also a small probability that the offsprings will be mutated. The offsprings created in each iteration are called a generation. Usually the size of the population is kept constant, so when a new generation is created some members of the current population will be replaced by members of the new generation. This replacement is typically performed on the basis of the fitness value, that is, the fitness value of each individual determines the probability of the individual being replaced. The algorithm is usually terminated after a certain number of iterations and the best individual in the last population returned as the best solution.

4.2.1 Original version of GA

The original version of the genetic algorithm is developed by Holland in 1960s and 1970s, see for example Reeves [14]. In the original version of GA one parent is selected on a fitness basis, that is, the probability of choosing a chromosome with a good fitness value is higher than for those with worse values. The other parent is chosen completely at random among all the individuals in the current population. Once the parents are selected they are mated and a pair of offsprings is generated using a crossover operation. In the crossover there is a single crossover point, which is chosen randomly. One offspring is then the combination of the part of the first parent preceding the crossover point and the second parents part behind the crossover point. The offsprings sibling is created in a vice versa manner. For an illustration of this crossover see figure 4.1. One of the members of the existing population is chosen randomly and replaced with one of the offsprings. This reproductive plan is repeated as many times as desired and the fittest solution in the last population is returned as a solution to the problem.

4.2.2 Other versions of GA

Several versions exist of the original genetic algorithm. Here some examples of possible variations will be given:

- Parents can be chosen in many other ways, such as selection based on ranking and tournament selection.

- More than one crossover point and their position chosen in other ways. It is even possible not to use specific crossover points but rather a more general crossover schema to decide which genes to inherit from each parent.
- More than one type of crossover and mutation operator used in the procedure.
- Each crossover can produce either one or two offsprings.
- Crossover/mutation probabilities varied in the iterative process.
- New generations can have as many individuals as desired. Usually the size of a new generation is in the range of one individual to the total number of individuals in each population, i.e. $[1, M]$.
- New generation and current population can be combined in a number of other ways, i.e. the individuals to be replaced are chosen in a non random manner.
- Fitness values can be chosen differently than purely by the objective function value, e.g. they can be mapped, scaled or ranked.
- Varying the population size during the iterative process.
- Different kinds of stopping criteria used, e.g. CPU time or no improvements for a certain number of iterations.

These possible variations will be discussed in more detail later in this chapter. A general pseudocode for the GA is given in algorithm 1.

Algorithm 1 Genetic algorithm

```
 1: Initial population constructed
 2: Initial population evaluated
 3: repeat
 4:   Select individuals from current population as parents
 5:   Offsprings created through crossover
 6:   Offsprings mutated with mutation probability
 7:   Offsprings evaluated
 8:   Population updated
 9: until Finished
10: Best individual returned
```

4.2.3 Convergence

In this section a discussion of the convergence aspects of the genetic algorithm will be presented. For a stochastic, iterative procedure such as the GA it is important that it is guaranteed to converge to one of the global optimum solutions if given enough time.

The GA can be constructed with or without mutation but if the GA only has a crossover operator and not mutation, it is not possible to prove that the method will reach one of the global optima or even local optima. Because crossover alone is not capable of reaching all the subspaces of the solution space the mutation is introduced to secure a (theoretical) full search of the solution space.

It is proven in Sait and Youssef [15] that the GA with mutation will converge to one of the global optimum solution when the GA runs for a large enough number of generations. Unfortunately, large meaning in most cases unpractical CPU-time.

Premature convergence

One of the problems that can rise in the genetic algorithm is called premature convergence. Premature convergence occurs if early in the iterative process the population becomes a set of many poor chromosomes and just one or two much better ones. Then the good chromosomes can rapidly take over and lead to a premature convergence to a poor local optimum early in the process, that is, selection has caused the search to narrow down too quickly. When this happens no new regions of the solution space are visited as the iterations progress and a great part of iterations go to waist.

4.2.4 Chromosome representation

An individual in the population represents a candidate solution to the problem. The individual is usually a string of numbers called a chromosome. It is very important to choose a good chromosome representation of the candidate solutions to the problem. A chromosome representation often used is a string of integers, either binary or positive integers. The order of the genes in the chromosome and the length of the chromosome are also important issues to consider.

1	0	1	1	1	0	1	0	0	0	0	1

Figure 4.2: An example of a binary chromosome representation.

Most of genetic algorithms assume a binary chromosome representation and an example of the binary chromosome representation can be seen in figure 4.2. GA has also been successful for problems requiring real integer values in their chromosome representation. Sometimes this is handled by mapping the real numbers onto a binary string. That can though present practical problems. The main problem is that values that are close in original space may be far apart in the binary-mapped space and vice versa. For this reason a non-binary coding may be preferable even though binary coding is straightforward.

Another chromosome representation worth mentioning is the sequence representation. The sequence representation is the most natural representation for many problems of

5	1	8	13	10	2	15
4	6	7				
16	12	14	11	3	17	
18	9					

Figure 4.3: An example of a two level chromosome representation. A two level representation can be thought of as a two dimensional matrix. Here the lines can e.g. represent four routes and the customers belonging to each route.

interest in OR such as for the travelling salesman problem and other problems where permutation is of importance. In the sequential representation the values of the genes indicate the order in which different nodes are visited, e.g. the chromosome $\boxed{2\ 1\ 3}$, indicates that node 2 is first visited, then node 1 and last node 3.

The chromosome representations described above can be extended to two levels. A two level chromosomes are for example used to represent a candidate solution to the multi-vehicle routing problem, see section 3.5, in which there is one line for each route and the genes indicate the customers in the route. In the two level chromosome representation it is possible use both binary and integer encoding. The two level representation can also be a sequence representation. An example of the two level chromosome representation can be seen in figure 4.3. Of course it is possible to have three or more levels of chromosome representation as well.

4.2.5 Schemata

A schema is a subset of similar chromosomes. Similar means that the chromosomes have an identical gene at the same position, a schema can for example be $\boxed{\ *\ 1\ *\ 1\ *\ }$ where * can be replaced by 0 or 1 for a binary chromosome representation. Then both $\boxed{\ 0\ 1\ 0\ 1\ 0\ }$ and $\boxed{\ 1\ 1\ 1\ 1\ 0\ }$ are a part of this schema. The chromosomes are obviously also members of several other schemata. In general if a chromosome is of length ℓ then it is an instance of 2^ℓ distinct schemata. Because at each position in the vector there can either be the chromosomes actual value or a *. In a non-binary coding the * in a schema is interpreted as a symbol that can be replaced by any *subset* of possible symbols. In theory a population of M members could contain $M2^\ell$ schemata.

Schemata are only used to illustrate certain properties of the GA and are not explicitly processed. That is, when implementing the genetic algorithm, no strings in the population represent partial solutions. All stings represent complete solutions.

The length of a schema is the distance between the first and last defined positions on the schema, i.e. non *. In the example above, $\boxed{\ *\ 1\ *\ 1\ *\ }$, the length of the schema is 2. The order of a schema is the number of defined positions, in the example above it is also 2. At the same time the fitness of a given chromosome is evaluated, information about the average fitness of each schemata of which the chromosome is an instance off is also gathered. The fitness ratio is the average fitness of a schema divided by the average fitness of the population.

Schema theorem
Using a reproductive plan where the parents are chosen based on their fitness and both crossover and mutation operators are used in which the probabilities of crossover and mutation are P_c and P_μ respectively. A schema S of order $ord(S)$, length $L(S)$ and has fitness ratio $f(S,t)$ at time t then the expected number of representatives of schema S at

time $t+1$ is given by:

$$E(S,t+1) \geq \{1 - \frac{P_c L(S)}{\ell - 1}(1 - P[S,t]) - P_\mu ord(S)\} f(S,t) N(S,t) \qquad (4.1)$$

where ℓ is the length of the chromosome, $N(S,t)$ is the number of representatives of schema S at time t and $P[S,t]$ is the probability that S will be represented in the population at time t.

Considerations of equation 4.1 show that the representation of S in the population will increase on the average provided that:

$$f(S,t) \geq 1 + \frac{L(S)}{\ell - 1} + P_\mu ord(S) \qquad (4.2)$$

This means that short low-order schemata have a larger chance than the longer high order once of representation.

4.2.6 Population size

The decision on population size is very important since the population size greatly influences the performance of the GA. In most implementations of the GA the population size is kept at a constant size M. Note that when choosing the population size and the population size is small, there is a great possibility of under-covering the solution space. When the solution space is under-covered it has not been investigated well enough. The solutions can be divided into groups based on for example values of some parameters. If there exists a group of solutions in which no solution has been investigated the solution space is said to be under-covered. When the population size is large, it is likely that representatives from a large number of solution groups are included and therefore the GA can perform a more informed search. Thus, large population size will discourage premature convergence to suboptimal solution but requires more evaluations per generation and may result in an unacceptable slow rate of convergence.

Some theoretical results imply that the optimal size for a population, in which there are binary coded strings, grows exponentially with the length of the string, ℓ. Other theoretical experiments suggest that some size between ℓ and 2ℓ is the optimal. Empirical results on the other hand show that population size as small as 10 to 50 usually is good enough for most problems, see Reeves [14] or Sait and Youssef [15].

The population size can also be allowed to vary in the iterative process which can be a good idea since the population size has a great effect on the total runtime of the algorithm. Experiments show that population fitness improves very rapidly in the first generations but improvement decelerates as the number of iterations increase. The reason for this is that improvements in the population are caused by fit individuals involved in crossover or mutation and in later generations the role of individuals in bad fitness becomes insignificant as a source of new fit individuals to the next generation. Therefore it seems reasonable to allow the populations size to progressively decrease with the number of generations. Such a decrease has been shown to cause a sizable reduction in runtime without any noticeable change in solution quality [15]. Another possibility is to vary the population size with the size of the data instances. For instance to have the population size proportional to the number of customers.

4.2.7 Initial population

The GA works with a population of individuals and therefore an initial solution constructor is needed. The initial solution is in most cases chosen randomly but some reports state [14] that if the algorithm starts with a solution of higher quality the GA tends to find better solutions faster. One possibility is to include solutions obtained from other heuristics into the initial population. The disadvantage is though that the possibility of premature convergence to a poor local optimum increases.

4.2.8 Stopping criteria

Usually the number of generations is used as a stopping criterion in the genetic algorithm. Other stopping criteria can also be used in the GA as in other iterative algorithms. The stopping criterion can for example be chosen to be a function of solution quality, the available runtime, no improvement for the last h iterations and so forth. The choice of the stopping criterion depends on the problem and must be chosen carefully in order to obtains a satisfactory solution.

4.2.9 Fitness calculations

The value of the fitness function in the genetic algorithm is required to be a positive number. It is often set to be the value of the objective function associated with each chromosome. This is however usually not a good idea since the populations tend to converge to a set of very similar chromosomes and the difference in the objective values of the chromosomes in such a set is very hard to detect. Other methods have been constructed in order to reduce the problems of using the raw objective value. An example of other methods to assign fitness values to individuals are for example scaling and relative order [14] and [15].

Scaling

Scaling is constructed so as to limit competition in the beginning and stimulate it at latter stages of the iterative process. A simple scaling procedure is to use the following transformation:

$$\Phi_S = c_1 v_S + c_2 \tag{4.3}$$

where Φ_S and v_S are respectively the fitness value and the objective function value for chromosome S. The constants c_1 and c_2 are obtained from the conditions:

$$\Phi_{mean} = v_{mean} \quad \text{and} \quad \Phi_{max} = \mu v_{mean} \tag{4.4}$$

where μ is added to ensure that the fittest member of a population will be chosen on a average μ times.

Ranking

When ranking is used it is the relative order of fitness of the chromosomes and not the actual, mapped or scaled objective value that is used. The individuals are sorted in ascending order of their fitness values and each of the chromosomes is assigned a new fitness value that corresponds to its number in the ascending order. Thereby the fitness function is dissociated from the underlying objective function and the selection depends on relative fitness rather than actual fitness values.

4.2.10 Selection mechanism

The selection mechanism is the method used to select parents to mate and to update the population in each iteration. Many types of selection mechanisms have been developed and in this section a description of three types of selection mechanism, namely stochastic sampling, selection based on ranking and tournament selection, and some possible variations are introduced.

Stochastic sampling

In the original version of GA one parent is chosen by means of a stochastic procedure, also called roulette wheel method or stochastic sampling with replacement, see figure 4.4, while the second parent is chosen purely at random. In the roulette wheel method the probability of choosing individuals is directly proportional to their fitness values. That is, the fitter an individual is, the greater chance of being selected. The new generation consists of one new offspring, which is set to replace a random member of the current population. The new population is the same as the old population with the exception of one individual. In this case populations are said to be overlapping. In order to control the overlapping of populations, a generation gap GAP is used. A generation gap is defined to be the proportion of existing population that is chosen to for reproduction and their offspring replaces a selected member of the existing population. In the original version of GA $GAP = M^{-1}$ (M is the population size), i.e. a single offspring replaces a random member of the population. $GAP = \frac{M}{M} = 1$, when as many offsprings as there are members of the current population are generated, the new generation replaces the old population completely. In this case there is no overlapping.

The genetic algorithm seems in general to work better when populations do not overlap. There is however the special case of incremental replacement (steady-state), $GAP = M^{-1}$, that has been very successful [14]. There are two main reasons for the success. The incremental replacement has certain advantages since it is easier to implement and each iteration is less time consuming. It is also easier to prevent occurrences of duplicates in each population using incremental replacement. Duplicates are unwanted since the same

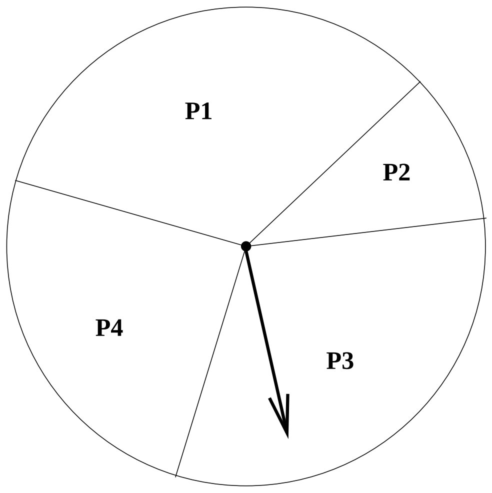

Figure 4.4: The roulette wheel method for choosing an individual in the population. Here there are four individuals and sizes of the individuals sections of the pie is directly proportional to their fitness values.

fitness value has to be evaluated twice and the solution space will not be explored as extensively. In addition duplicates distort the selection process by increasing the duplicate chromosomes probability of being selected.

Another version of the stochastic selection mechanism has been constructed, in which it is made sure that the best member of the current population will survive to the next generation. That is, the best member is forced to become a member of the next population. This is for example performed in the incremental replacement when the member to be replaced is only chosen randomly among those that have fitness value below average. It is also possible to define a section, proportional to the population size, of the individuals that is allowed to select for replacement. This kind of selection is of great interest for optimization problems, since in optimization problems we are looking for the global optimum solution. This method makes sure that the best candidate solution always survives to the next population, i.e. is never replaced by an offspring and therefore never lost.

The selection of parents to mate is subject to sampling errors, which sometimes leads to a serious difference between the actual and expected number of times a chromosome is used in mating. In order to account for this, the expected value model is constructed. In the model, chromosomes are forced to become parents as often as their expected frequencies predicted by their fitness values demand. This method is used without replacement and seems to out-perform the conventional approach of selection with replacement.

Selection based on ranking

Selection methods, like the stochastic sampling, that rely on the raw or scaled value of the objective function to select parents and members of new population, work only for positive fitness values and can be problematic, since they can cause premature convergence. A key to a good GA performance is to maintain an adequate selective pressure[2] on all the individuals by means of an appropriate relative fitness measure. This can be accomplished by ranking, see section 4.2.9. Here potential parents are chosen based on the probability distribution:

$$P(S) = \frac{2S}{M(M+1)} \tag{4.5}$$

where S is the Sth chromosome in the ascending order and the population size is M. The best chromosome ($S = M$) has the chance of $\frac{2}{M+1}$ of being selected which is roughly twice the chances of choosing the median, that has a chance of $\frac{1}{M}$. Ranking has been able to produce better results [17] than stochastic selection.

Tournament selection

Tournament selection combines both selection and ranking mechanisms. In this schema the population is treated as a permuted list of M chromosomes. The index numbers of the chromosomes are randomly permutated. The population is divided into successive sub-groups of $\tau \geq 2$ chromosomes. The chromosome in each group are compared and the fittest chromosome in each sub-group chosen as parent. The list is randomly permutated again and the whole procedure repeated until M parents have been chosen. Each parent is then mated with a second parent randomly chosen from the whole population. In this procedure the best chromosome is selected τ times, the worst chromosome never and the median chromosome is chosen once on the average. The special case of $\tau = 2$ has similar effects as ranking but there is no need to keep an ordered list of the chromosomes.

4.2.11 Modifying operators

In many cases the simple crossover operator (as is shown in figure 4.1) has proved to be extremely effective and mutation encourages population diversity that helps the procedure to escape from local optimum regions. The procedure has high tolerance level regarding the rate of mutation but if the rate gets too high it could get in the way of the crossover operations.

The modifying operators also depend on the chromosome representation used in the problem. There are for example problems when using the simple crossover operators in sequential chromosome representation, since it could for example result in chromosomes with many genes of the same values. In figure 4.5 both the offsprings represent infeasible tours as there are duplicate genes and other genes are missing. The genes can for example represent customer numbers and then the above mentioned errors would mean that there are some customers missing from the offsprings solution, while other customers are to be visited twice (or more). Of course it is possible to use this operator and then use a fix-up method to eliminate duplicates and insert missing customers in the offsprings chromosomes. It might however be wise to investigate if other crossover methods are better

[2]Selective pressure is defined as the probability of the best individual being selected compared to the average probability of selection of all individuals.

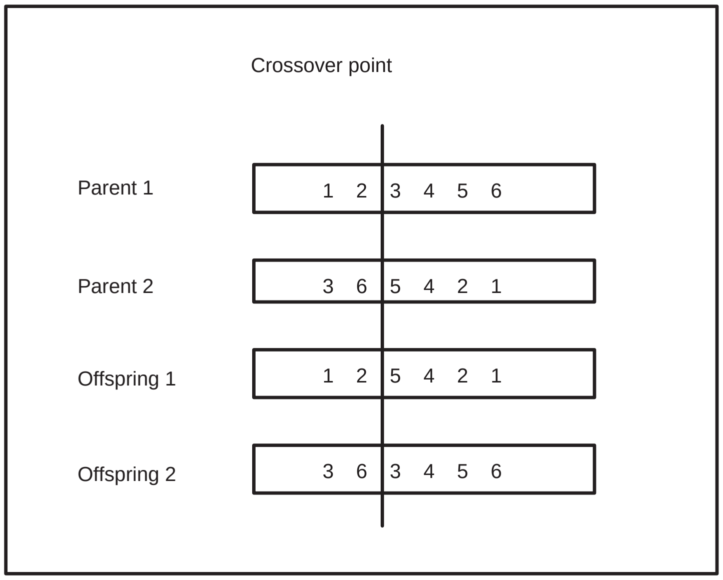

Figure 4.5: Example of how the simple crossover works on a sequential chromosome representation. The offsprings get a section before the crossover point from one parents chromosome and the section after the crossover point from the second parent.

suited for this problem.

Several crossover operators have been developed for solving the crossover problem described above. For other kinds of chromosome representation, where the simple crossover and mutation operators have been very successful, more advanced operators have also been developed. In this section some of the modifying operators will be described.

String-of-change crossover

Often, especially at later stages of the iterative process of the GA, the individuals in the population converge to such an extent that the crossover has little effect. For example the parents $\boxed{1\ 0\ 0\ 0\ 0\ 1}$ and $\boxed{1\ 0\ 0\ 1\ 0\ 0}$ will fail to produce offsprings that differ from themselves, if the crossover point is in any of the first three positions. The idea of the string-of-change crossover is to find a crossover point that makes sure that the offsprings differ from their parents. In the method an XOR string is computed where the 0s represent same element in same position of both parents and the 1s different elements. For example the XOR string for the example above is: $\boxed{0\ 0\ 0\ 1\ 0\ 1}$. The crossover point is then only allowed to be between the outermost 1s of the XOR string, that is, in any of the last two positions in this example.

Uniform crossover

The uniform crossover is a generalization of the simple crossover. In the uniform crossover the crossover procedure is performed based on a binary string (template) that indicates which elements are to be taken from which parent. For example the template $\boxed{1\ 1\ 0\ 0\ 0\ 0}$ says that the first two elements are to be taken from one parent and the

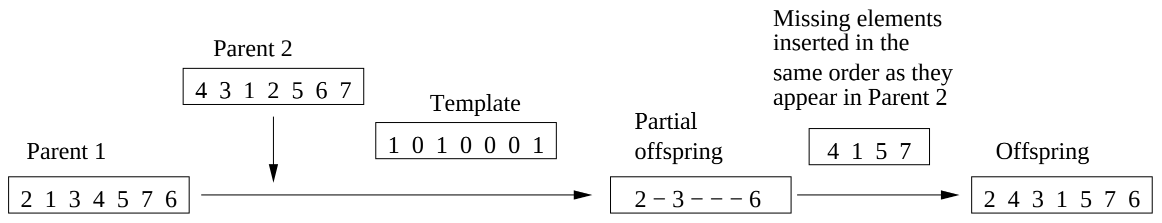

Figure 4.6: An example of the uniform order-based crossover. Two parents are chosen and the genes from parent 1 are inherited by the offspring in places where the template has a value of 1. The rest of the genes are inserted into the offspring in the same order as they appear in parent 2.

last four from the other parent. This is the way the simple crossover works in and it can be generalized by allowing the pattern of 0s and 1s to be generated stochastically using a Bernoulli distribution. For example the template $\boxed{1\ 0\ 1\ 0\ 1\ 1}$ implies that the 2nd and 4th elements are taken from one parent while the other elements are taken from a second parent.

The advantages over the simple crossover is that the procedure is indifferent to the length of schema, that is all schemata of given order have the same chance of being disrupted. This operator gives the possibility of more varied offsprings to be produced and therefore a better coverage of the solution space.

Inversion

In the simple crossover it is assumed that there is no order-relationship between adjacent genes. If this is not the case re-ordering operators can be of importance. Inversion is an operator that takes into account the order relationships between genes. A section of the chromosome is cut out and re-inserted again in the reverse order. For example $\boxed{2\ 7\ 5\ |\ 6\ 1\ 3\ |\ 4}$ becomes $\boxed{5\ 7\ 2\ 3\ 1\ 6\ 4}$. Inversion does not have re-combinative power of crossover and has not been found to be significantly useful.

C1 operator

The C1 operator is specially adapted for sequential chromosomes. The C1 operator chooses a random crossover point. Then takes the elements preceding the crossover point of one parent and fills up the chromosome by taking in order each legitimate elements from the second parent. For example if parent 1 is $\boxed{2\ 1\ |\ 3\ 4\ 5\ 6\ 7}$ and parent 2 is $\boxed{4\ 3\ |\ 1\ 2\ 5\ 7\ 6}$ then the offsprings become $\boxed{2\ 1\ 4\ 3\ 5\ 7\ 6}$ and $\boxed{4\ 3\ 2\ 1\ 5\ 6\ 7}$. This procedure preserves the absolute gene position of one parent and the relative order of genes of the other parent and the offsprings represent feasible solutions.

Uniform order-based crossover

The uniform order-based crossover is also adapted for sequential chromosomes and it can be said to be a generalization of the C1 operator. This method combines ideas from the

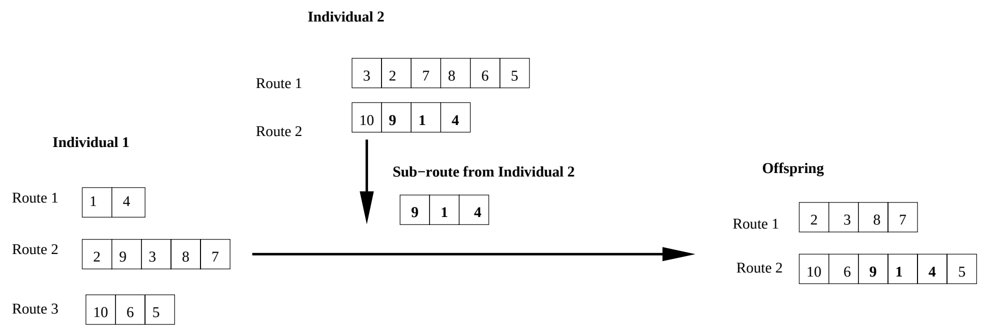

Figure 4.7: An example of a GVR-crossover. An offspring is created by selecting a section of parent 2, inserting it into parent 1 and removing duplicates. It is assumed customer 6 is closest to customer 9.

C1 operator, about using legitimate elements to fill up the chromosome, and the uniform crossover, about allowing stochastic order of elements in the templates. Here the 1's in the template define elements copied from the first parent, while the other elements are copied from the second parent in the order they appear in the chromosome. The second offspring is created in the same manner. For an example of how the uniform order-based crossover works see figure 4.6.

GVR-crossover

In the GVR-crossover an offspring is created by inserting a fragment of genetic material from one parent into one chromosome of the other parent. In the VRP this could for example represent the insertion of a sub-route from one parent into one route of the second parent. In that case the placement of the insertion is directly behind a customer that is not a part of the sub-route and is closest to the first customer in the sub-route. Afterwards duplicates in other chromosomes are removed. This example of a GVR-crossover is pictured in figure 4.7. This figure is also presented in chapter 3 but is represented here. The crossover is capable of reducing the number of routes, changing the order of customers in routes and reallocating customers to routes.

Mutation

Mutation is the random modification of some element or elements of a chromosome. In the case of a sequential chromosome representation this may not be possible and the mutation needs to be modified. Here three modifications will be described.

- **Exchange mutation**
  Two randomly chosen elements of the permutation are interchanged.

- **Shift mutation**
  A randomly chosen element is moved a random number of places to the left or right. The shift mutation has proved to be superior to the exchange mutation [14].

- **Scramble sublist mutation**
  Two points on the string are chosen at random and the section in between the

two points is randomly permuted, i.e. scrambled, the elements between these two points. Some times it is necessary to limit the length of the portion, which is to be scrambled. The scramble sublist mutation is considered to be superior to the exchange mutation [14].

4.3 Modified space-time nearest neighbour heuristic

The modified space-time nearest neighbour heuristic will be described in this section. The heuristic is developed by Baugh et al. [2] and is used in an extended version for building routes. The extensions that are needed in order to keep the constraints for the DARP formulated will be described in the next chapter.

The modified space-time nearest neighbour heuristic is a greedy algorithm based on the space-time nearest neighbour heuristic. The heuristic is constructed to develop a good route given customers with pickup and drop off locations along with time windows for both locations.

The heuristic starts by selecting the first customer in the route. The first customer is the customer with the earliest pickup time. The pickup location of the first customer is the first stop in the route. To find the next stop four candidate stops are considered. The candidate stops are the four stops that are closest, in space and time, to the first customers pickup location. The next stop is chosen as the cheapest stop of the four candidate stops. The cost of a stop is evaluated as the cost of the possible next three succeeding moves after that stop. This procedure is repeated until a route consisting of all the customers pickup and drop off locations has been generated.

The selection of the four stops, that are taken into consideration as the next stop in the route, is performed as follows. First the drop off locations of the customers already in the vehicle are considered and the one closest to the current stop in space and time is chosen. If there is an empty seat in the vehicle then the pickup locations of customers that have not yet been serviced are considered. If there exists such a pickup location that is closer to the current stop in space and time than the closest drop off location (if any), then that stop is chosen as the first of the four stops, otherwise the drop off location is set as the first. This procedure is repeated three times, which results in four possible next stops.

The closeness of two stops is measured using a space-time separation between the stops. The space-time separation between two stops is quantified by a weighted sum of travel time between the stops and the time window violation at the latter stop. The time window violation is positive if the latest time at which a destination stop can be reached precedes its time window. The time window violation is negative if the earliest time at which a destination stop can be visited exceeds its time window.

The cost of a move between two locations is set to be the weighted sum of travel time and the absolute amount by which the time window is violated at the latter location.

In the algorithm the precedence constraint is not violated. The capacity constraints on the vehicles are considered as hard constraints, while the time window constraints are considered to be soft. Other constraints present in the dial-a-ride problem are not a part of this heuristic algorithm in which it is also not possible to reject customers.

A pseudocode for the modified space-time nearest neighbour heuristic is given in algorithm 2.

Algorithm 2 The modified space-time nearest neighbour heuristic

```
’ Global data
 1: $C \leftarrow$ SetofCustomers
 2: $CS \leftarrow$ SetofCustomersServiced
 3: $CV \leftarrow$ SetofCustomersinVehicle
 4: $CN \leftarrow$ SetofCustomersNotServiced
 5: $Speed$
    ’ Route
 6: $FirstCustomer \leftarrow c \in C$ with the earliest pickup time
 7: $Ctime \leftarrow FirstCustomer$ pickup time
 8: $CurrentNode \leftarrow FirstCustomer$
 9: $CS \leftarrow \emptyset$
10: $CV \leftarrow \{FirstCustomer\}$
11: $CN \leftarrow C - CV$
12: while ($CS \neq C$) do
13:     $N4 \leftarrow$ 4 nodes space-time closest to $FirstCustomer$
14:     Choose cheapest $n \in N4$ as $NextNode$
15:     Visit($NextNode$)
16:     $Ttime \leftarrow$ distance from $CurrentNode$ to $NextNode$ divided by $Speed$
17:     $Ctime \leftarrow Ctime + Ttime$
18:     if($Ctime <$ earliest time window at $NextNode$) do
19:         $Ctime \leftarrow$ earliest time window at $NextNode$)
20:     end if
21:     $CurrentNode \leftarrow NextNode$
22: end while
    ’Visit($NextNode$)
23: Update global data
```

Chapter 5

Implementation

In this chapter a description of how the chosen solution method, cluster-first, route-second, is used to solve the formulated dial-a-ride problem will be given. The chapter starts by an overview of the solution method. Next a detailed description of the formulated dial-a-ride problem and the necessary relaxations are presented. A relaxation of an optimization problem can e.g. be by converting hard constraints to soft constraints. Thereafter a description of the implementation of the genetic algorithm used for clustering is presented. The decisions on the structure of the genetic algorithm are discussed. These decisions are choice of chromosome representation, population size, initial population, stopping criteria, fitness calculation, selection mechanism and modifying operators. Next a description of the implementation of the modified space-time heuristic is presented along with the extensions needed to solve the formulated problem. Finally an overview is given of the initial heuristic and improvements proposed.

5.1 Overview of the solution method

The implementation of the formulated dial-a-ride problem is divided into two parts, clustering and routing. In the clustering part as many groups of customers as there are available transportation vehicles are created. A cluster is allowed only to contain the origin and destination depot and no customers, i.e. the vehicle is not used. However, the model does not include minimization of the number of vehicles. Each customer can only belong to one group and only one group can be assigned to each vehicle. The clustering of customers is solved using the genetic algorithm. When all the customers have been grouped, a route for each vehicle is constructed. The routing involves deciding the order of the stops of the vehicle along with the time table for the vehicle, that is, in which order the customers belonging to that particular group are to be picked up and dropped off and at what time. The routing is solved using an extended version of the modified space-time nearest neighbour heuristic developed by Baugh et al. [2] It will be referred to as the Baugh routing-heuristic in this chapter. The cost of the cluster is then calculated based on the particular route developed for the cluster. The cost calculations are a part of the routing heuristic.

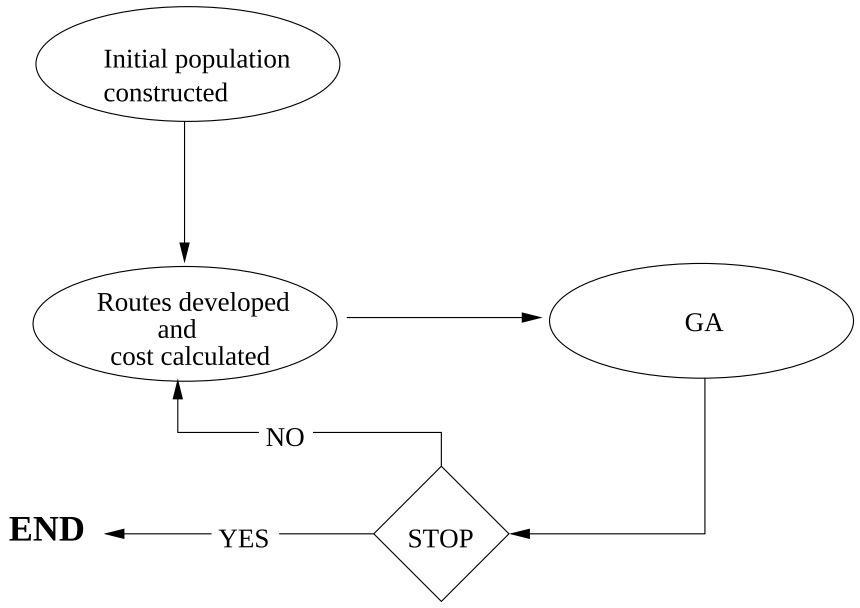
Initial population constructed
Routes developed and cost calculated
GA
NO
END
YES
STOP

Figure 5.1: An overview of the solution process.

The genetic algorithm works with a set of solutions, called a population, and uses them to create new solutions. Therefore an initial population generator is needed before the GA can start its iterative procedure of finding a good solution to the problem. The solution method starts by creating an initial population, in which all the customers are randomly clustered. Next routes are generated for the initial clusters and costs calculated using the Baugh routing-heuristic. The costs are returned to the GA and used to create new solutions. Routes are developed for the clusters in the new solutions and costs evaluated. This procedure is continued until the termination condition is reached. The best solution in the last population is returned as the solution to the problem. In figure 5.1 a schematic overview of the solution method is given.

5.2 Relaxation

The dial-a-ride problem, as presented in chapter 2, is difficult to solve directly. One way of simplifying the problem is to drop some of the constraints in the problem. The resulting problem is a relaxation of the original problem. The constraints to be dropped are added to the objective function with an adjustable multiplier. These new terms in the objective function represent the penalty for violating the relaxed constraints. When a constraint has been dropped and then added to the objective function in this manner it is called a soft constraint. The original constraint in this case is said to be hard. Models with soft constraints can be considered more general than models with hard constraints, since soft constraints can model hard constraints by imposing very large penalties for violating the constraints. Algorithms based on soft constraints are also capable of finding solutions in cases are algorithms based on hard constraints would fail to find any solutions to the problem. That will happen when there are no solutions at all to the problem which fulfil all the hard constraints or the algorithm is stopped before finding any legal solutions.

The constraints in the dial-a-ride problem can both be represented as hard and soft. The decision on whether or not a constraint is allowed to be converted to a soft constraint is based on an analysis of the underlying problem. The formulated problem in this thesis is purely theoretical but takes into account practical issues from the Danish transportation operations. Lets look at the constraints in the problem one at a time.

- *Depot constraint* must be hard, otherwise each tour will end at the last customers drop off location. That is not practical since there is a bus driver in each vehicle who demands to start and end his tour at one of the depots. It is either not desirable to leave the vehicle unattended somewhere in the city.

- *Routing constraint* is also kept as a hard constraint, because it is necessary to visit all the pickup and drop off locations of the customers, since it is not allowed to reject customers in the formulated problem.

- *The maximum route duration constraint* is, on the other hand, converted to a soft constraint. The upper limit on the route duration in this project represents the length of the workday of the drivers and it is possible that the drivers can work a little longer if needed. One could imagine that overtime pay or compensation by days off at another time are a part of the job description of the drivers.

- *The constraint on maximum customer ride times* is also converted to a soft constraint. The upper limit on maximum ride times is a constant for each customer. The reason for relaxing this constraint is simply making implementation easier. It also suits the transportation operator, which usually sets this upper limit.

- It is not possible to relax *the precedence constraint*, since it can produce infeasible solutions. That is, it is physically impossible to drop a customer off before the customer is picked up.

- *The time windows constraints* can be relaxed. The advantage of converting hard time windows to soft time windows is that in practice time windows are usually soft, in the sense that there is a limit up to which a service provider will adhere to the time windows. The soft time windows are also useful in evaluating the tradeoffs between service requirements and cost requirements. Solutions with soft time windows indicate the degree of violation allowing penalty methods to distinguish between a given pair of infeasible solutions in attempting to find a feasible region. The time windows are constructed based on the desired pickup or drop off time given by the customer. The time windows constraints will be converted to soft constraints, i.e. it is allowed to adhere to the desired time given by the customer at an additional cost.

- *The capacity constraints* can be relaxed if desired. That is however problematic if the capacity of all the vehicles fixed and there are no means of adding an extra customer to the vehicles. The capacity constraint is therefore set to be hard.

For the constraints it has been decided to relax, i.e. time windows, customer ride times, and route duration constraints, penalty factors are added to the objective function along with adjustable multipliers.

An additional modification is made to the model presented in chapter 2, which is that the cost on number of vehicles used is dropped, since it has been decided to have a constant number of vehicles available. Because even though a vehicle is not used it is still available and the fixed cost cannot be avoided. This is also usually the case in the Danish transportation operations.

The relaxed mathematical model is presented on the next page. The notation from the mathematical model in chapter 2 is used.

Minimize

$$\begin{aligned} & w_1 \sum_{k\in V}\sum_{i,j\in A} t_{i,j}x^k_{i,j} + w_{3'}\sum_{k\in V}\sum_{i\in P}(T^k_{n+i} - s_i - T^k_i - t_{i,n+i}) + w_4 \sum_{k\in V}\sum_{i\in N} W^k_i(L^k_i - l_i) + \\ & w_5 \sum_{k\in V}(T^k_{d(k)} - T^k_{o(k)}) + w_6 \sum_{k\in V}\sum_{i\in A}\max(0, a_i - T^k_i, T^k_i - b_i) + \\ & w_7 \sum_{k\in V}\sum_{i\in P}\max(0, (T^k_{n+i} + T^k_i) - u_i) + w_8 \sum_{k\in V}\max(0, (T^k_{d(k)} - T^k_{o(k)}) - r^k) \end{aligned} \tag{5.1}$$

Subject to

$$\sum_{k\in V}\sum_{j\in P\cup d(k)} x^k_{o(k),j} = m \tag{5.2}$$

$$\sum_{k\in V}\sum_{i\in D\cup o(k)} x^k_{i,d(k)} = m \tag{5.3}$$

$$\sum_{j\in A} x^k_{i,j} - \sum_{j\in A} x^k_{j,i} = 0 \qquad \forall k\in V, i\in N \tag{5.4}$$

$$\sum_{k\in V}\sum_{j\in N} x^k_{i,j} = 1 \qquad \forall i\in P \tag{5.5}$$

$$\sum_{j\in N} x^k_{i,j} - \sum_{j\in N} x^k_{j,n+i} = 0 \qquad \forall k\in V, i\in P \tag{5.6}$$

$$x^k_{i,j}(T^k_i + s_i + t_{i,j} + W^k_j - T^k_j) \le 0 \qquad \forall k\in V, i,j\in A \tag{5.7}$$

$$T^k_i + s_i + t_{i,n+i} + W^k_j - T^k_{i+n} \le 0 \qquad \forall k\in V, i\in P \tag{5.8}$$

$$x^k_{i,j}(L^k_i + l_j - L^k_j) = 0 \qquad \forall k\in V, i,j\in A \tag{5.9}$$

$$l_i \le L^k_i \le C^k \qquad \forall k\in V, i\in P \tag{5.10}$$

$$L^k_{o(k)} = L^k_{d(k)} = 0 \qquad \forall k\in V \tag{5.11}$$

$$x^k_{i,j}\in\{0,1\} \qquad \forall k\in K, i,j\in A \tag{5.12}$$

5.2.1 Effect of arrival time to destination

The actual arrival time of a customer to the destination location is part of many factors in the objective function. The factors that can be affected are: The time window violation excess ride time, ride time violation, route duration, route duration violation, and waiting time.

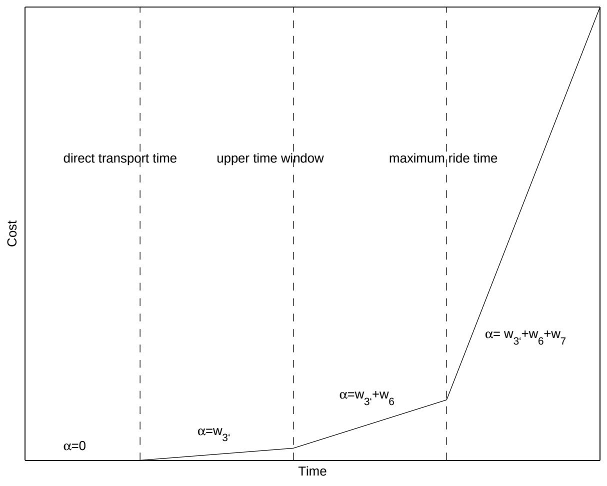

Figure 5.2: The influence of the arrival time on cost. The slope (α) of the cost increases in steps depending on the arrival time.

If the customer arrives on time to his destination within the time windows, the cost contributed by that customers arrival is, if any, the excess ride time. On the other hand, if he is late the time that passes from the upper bound of the time window to actual arrival is both the excess ride time and punishment for every minute the arrival time exceeds the upper time limit. If the ride time of the customer also exceeds the maximum ride time a penalty for that is also added to both the cost of excess ride time and time window violation. Above we are assuming that the maximum ride time for the customer plus the pickup time is higher than the upper bound on the drop off time window. This case is presented in figure 5.2. So the cost of delivering a customer late increases in steps depending on these three constraints and how late the customer is delivered.

If the vehicle arrives too early to a location it has to wait. Two cases can arise; no customers are present in the vehicle, or one or more customer sits in the vehicle. In the first case the minutes the vehicle has to wait adds a penalty to the cost for violating the time windows, the route duration increases and possibly a route duration violation occurs, if the maximum route duration is exceeded. The reason for having a penalty for a time window violation even though the vehicle is empty is to have the implementation of the time window violation uniformed. If there are customers present in the vehicle the waiting minutes will add to the cost in several ways: A punishment for time window violation adds to cost, the waiting time multiplied by the number of customers present in the halting vehicle also adds to cost, excess ride time for the customers, route duration increases and possibly ride time or route duration violations as well.

5.3 The initial heuristic - GA1

In this section the initial heuristic, denoted GA1, used for solving the formulated DARP is described. The section is divided into two main parts which are the same as the two main parts of GA1, i.e. clustering and routing. In the part about clustering a discussion

	depot	customer 1	customer 2	customer 3	customer 4	customer 5	customer6	customer 7	customer 8	customer 9	customer 10	customer 11	customer 12	customer 13	customer 14	customer 15	customer 16
Route 1	1	0	1	0	1	0	0	0	1	1	1	0	0	1	0	1	0
Route 2	1	1	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0
Route 3	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0
Route 4	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1

Figure 5.3: An example of the binary chromosome representation used in the solution method. There are as many columns as customers and depots and as many rows as routes. The number of routes equals the number of available vehicles. Each customer must be assigned to exactly one route and each route has to include the depot.

about the decisions on the structure of the genetic algorithm is presented. In the part on routing the necessary extensions needed for the Baugh routing-heuristic are presented.

5.3.1 Clustering using GA

The construction of the GA involves taking decisions about which chromosome representation that will best fit the solution to problem, which population size is adequate, how the initial population should be generated, what stopping criteria to use, how the fitness calculations should be made, what kind of selection mechanism is best to use, and which modifying operators to use.

Chromosome representation

In order for an individual in the population to represent a solution to the problem of allocating customers to vehicles it is decided to use a two level binary chromosome representation. This representation is a binary version of the chromosome representation introduced by Pereira at al. [13].

In this representation there are as many lines as there are available vehicles and there are as many columns as there are customers and depots. A 1 at any position $[g1, g2]$ indicates that the customer/depot represented by column $g1$ is allocated to the vehicle represented by line $g2$. In this chromosome representation it is very easy to verify that each customer is allocated to exactly one vehicle. In the formulated dial-a-ride problem there is only one depot and it is also very easy to verify that all the vehicles routed include the depot. The verification is performed by adding up all the columns in the chromosome and checking whether or not the sum equals the number of vehicles for the first column (all the routes include the depot) and to 1 for the other columns (each customer is allocated to exactly one vehicle). A chromosome that fulfills these constraints, the constraint on each customer allocation to exactly one vehicle and the depot constraints, is a legal solution to the clustering part of the problem.

In figure 5.3 there is an example of what the chromosome representation looks like when there are four vehicles available, one depot and 16 customers. As we can see each customer is allocated to exactly one vehicle. The routes can be constructed in many ways for example route 1 could be constructed in the following way:

depot - 9.1 - 15.1 - 15.2 - 2.1 - 9.2 - 8.1 - 10.1 - 8.2 - 2.2 - 10.2 - 13.1 - 13.2 - 4.1 - 4.2 - depot

where the $i.1$ is the pickup location of customer i and $i.2$ is the drop off location of customer i, for all i in route 1.

Population size

The size of the population greatly influences the performance of the GA. If the population size is too small it results in a high possibility that the solution space will be under-covered, while a too large population will be a burden on the computational time and may lead to unacceptable slow rate of convergence.

In the section about population size for the genetic algorithm (see section 4.2.6) no specific guidelines about decisions on the population size are given but instead the different findings of some theoretical results are given. Some of which indicate that a population size between the length of the string used in the chromosome representation and the double length is optimal, while empirical results on the other hand indicate that a populations size as small as 30, independent on the problem size, works well for most problems.

It is decided to use the conventional method of a constant population size and perform some preliminary tests to decide how big the population size has to be for the problems that are to be solved. Tests of different sizes of the population will be presented in chapter 6.

Initial population

The initial population is created randomly. The consideration kept in mind in the initial population generator is that each customer must be allocated to exactly one vehicle. This is easily checked in the chromosome representation by adding up the columns and make sure that this equals exactly one for all the customers. Another consideration is the depots. In the problem formulation of this project (section 2.6) it is decided to use only one depot so the initial solution must have the depot presented in all clusters. That is the summation of the depot column must equal the number of vehicles available. All the initial solutions represent therefore a legal solution to the clustering part of the problem.

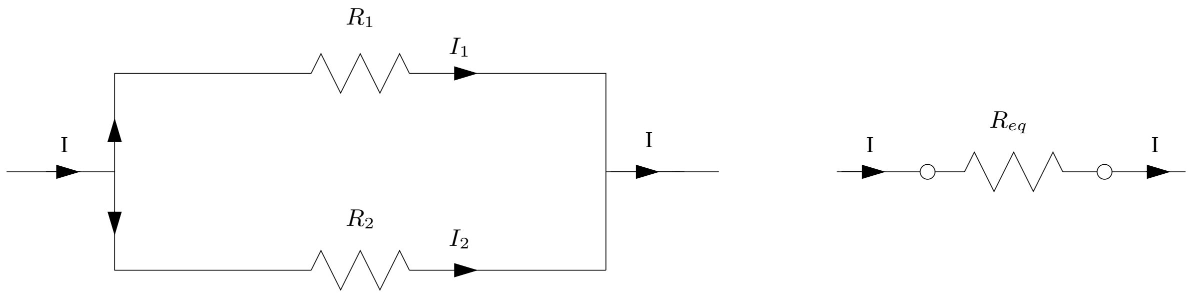

Figure 5.4: Two resistors in parallel (R_1 and R_2) have equivalent resistance given by $\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2}$ and I is the total current, I_1 and I_2 are the currents going through resistors R_1 and R_2 respectively.

Stopping criteria

It is possible to choose from different stopping criteria, such as number of iterations, function of solution quality, the available runtime and no improvements for the last specific number of iterations. The most common is to use the number of iterations for stopping criteria and that will also bee the case in GA1. The main reason for that choice, besides the wide spread use of it, is that it is easily implemented, the number of iterations can easily be altered and are proportional to the runtime. The actual number of iterations needed is hard to decide, but in chapter 6 the influence on solution quality for three different number of iterations will be investigated.

Fitness calculations

In the initial algorithm GA1 the fitness of an individual in the population is purely the value of the objective function for that particular solution. This is not recommended to [14] but it is easiest to implement and gives possibilities to investigate the effect of using other methods.

The fitness value of a chromosome that represents solution S is:

$$\Phi(S) = v_S = f_{cost}(S) + f_{penalty}(S) \tag{5.13}$$

where v_S is the objective function value for a solution S. $f_{cost}(S)$ represents the four first segments in the objective function 5.1 of the relaxed model and $f_{penalty}(S)$ represents the total punishment for violating the relaxed constraints, i.e. the last three segments of the objective function 5.1 in the relaxed model. For a feasible route, in view of the problem before relaxation, the $f_{penalty}(S)$ will equal to zero but for an infeasible route to the unrelaxed problem the value of $f_{penalty}(S)$ will be larger than zero, see section 5.2. Even though the problem has been relaxed, at this point it is still desirable to obtain solutions that fulfill the relaxed constraints or violate the relaxed constraints as little as possible. Therefore, in the implementation of the objective function, the segments $f_{penalty}$ represents get a large weight, i.e. w_6, w_7 and w_8 are set to high values.

Selection mechanism

In the initial algorithm GA1 two individuals are chosen to produce one offspring. One parent is chosen by means of a stochastic procedure, while the second one is chosen randomly. The stochastic procedure also called the roulette wheel method, see section 4.2.10, gives each individual in the current population a probability of being chosen as parent proportional to the fitness value of the individual, i.e. if the individual is in good fitness (represents a low cost solution) the probability of choosing that individual is high, while the probability of choosing an individual in bad fitness is low. The only problem encountered using this method is that it is designed for solving a maximization problem, i.e. the higher fitness value an individual has the more probable it is to chose that particular individual. This is the opposite to solving the dial-a-ride problem, which is a minimization problem. We are looking for the cheapest solution and therefore the lowest fitness value. This problem could have been solved by scaling or ranking the objective value in the fitness calculations but it is decided to use the raw objective value in the initial heuristic. The problem is to give solutions with low fitness values high probabilities and vice versa and making sure that the probabilities add up to 1. This problem resembles a problem encountered in electrical physics where the task is to find the current through a set of parallel resistors, see figure 5.4. First it is necessary to calculate the equivalent resistance for the parallel resistors, i.e. how big a resistor is needed if the parallel resistors are to be replaced by a single resistor. Using Ohms law, junction rule and that the resistors have the same potential difference across them, the equivalent resistance is given by:

$$\frac{1}{R_{eq}} = \sum_i \frac{1}{R_i} \tag{5.14}$$

where R_i is the equivalent resistance and R_i is the resistance in resistor i. The current through each resistor is then proportional to the current that enters the parallel connection (indicated by I in figure 5.4) and the equivalent resistance divided by the current in the resistor, i.e.

$$I_i = I\frac{R_{eq}}{R_i} \quad \forall i \tag{5.15}$$

where I_i is the current through resistance i. Since the same current enters and leaves the parallel resistor the following is true:

$$1 = \sum_i \frac{R_{eq}}{R_i} \tag{5.16}$$

This can be extended to the problem of constructing a roulette wheel for minimization problem. Then the resistance is substituted with the objective function value and the proportional current through each resistance (I_i/I) with the probability of using a solution. That is:

$$\frac{1}{v_{eq}} = \sum_S \frac{1}{v_S} \tag{5.17}$$

where v_S is the objective value for solution S. The probability of selecting individual S is then given by:

$$P(S) = \frac{v_{eq}}{v_S} \quad \forall S \tag{5.18}$$

The roulette wheel method in which the selection probabilities are calculated using equation 5.18 is used for selecting parent 1 in GA1.

In each iteration one offspring is created, which replaces one random member that belongs to the Z portion of the current population that have the worst fitness values, i.e. represent solutions with high cost. Z is set to be proportional to the population size and three different values for Z will be tested in the next chapter. Using this method for updating the population, called incremental replacement, the best member of a current population is guaranteed to survive to the next population. The best solution encountered so far will therefore never be lost and is allowed to influence the development of future generations.

Modifying operators

The main purpose of the crossover operator is to generate better populations as the algorithm progresses. The mutation operator helps increasing diversity of the population and coverage of the solution space. It is necessary to have both mutation and crossover present in the genetic algorithm, otherwise it is impossible to prove the convergence of the GA.

The modifying operators in this problem must be capable off reallocating customers from one cluster to another, in order for the heuristic to be able to investigate the entire solution space. The modifying operators must also generate legal solutions to the clustering problem. In a legal solution to the clustering problem, each customer must be allocated to exactly one vehicle and the depot must be a part of every route.

- **Crossover**
  The crossover used in the algorithm is a version of the crossover described by Pereira et al. [13], which is described in section 4.2.11. In this crossover, one line, i.e. one cluster, is chosen at random from both parents and a random binary template is created. The template is used as a recipe for one line in the offspring, where a 1 in the template indicates that the gene is to be taken from parent 2 and the rest from parent 1. The offspring consist of the new line while the other lines are duplicates from parent 1. When constructing a new solution in this manner it does not necessarily represent a legal solution. Therefore it is checked if a customer exists that is assigned to more than one vehicle or no vehicle at all. If such a customer is found, a cluster is chosen randomly and either the customer is added or deleted from that cluster depending on whether the customer is allocated too often or never. If the randomly chosen cluster is the cluster created in the crossover a new cluster is randomly chosen. This procedure is repeated until all the customers are allocated to exactly one vehicle. It is not necessary to verify that the depot is present in every cluster since the parents represent a legal clustering solution. In each iteration one crossover is performed resulting in one offspring.

- **Mutation**
  In the initial heuristic a simple mutation operator is used. The mutation operator moves one random customer from its current cluster to another random cluster. It is not possible to generate illegal solutions here using this mutation so no verification or correction procedure is needed after mutation. The offspring that has been created in the crossover can be subjected to mutation with a certain probability, called the

mutation probability, P_μ. It is also possible to apply mutation on other individuals than the new offspring. The normal procedure is to apply mutation with a fixed mutation probability to offsprings, which will be performed here as well.

5.3.2 Routing strategy

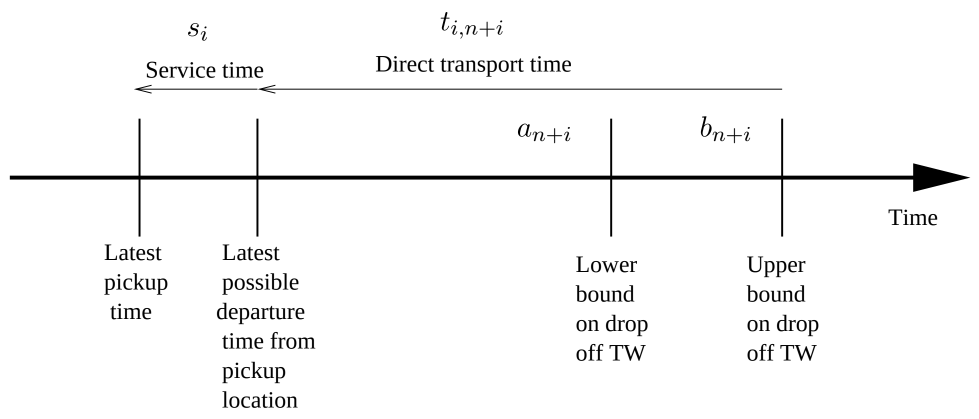
s_i
Service time
$t_{i,n+i}$
Direct transport time
a_{n+i}
b_{n+i}
Time
Latest pickup time
Latest possible departure time from pickup location
Lower bound on drop off TW
Upper bound on drop off TW

Figure 5.5: Illustration of the latest pickup time calculations for an outbound customer. The latest pickup time is calculated backwards in time. The direct transportation time from pickup to drop of location and the service time at the pickup location are subtracted from the upper bound on the drop off time window.

Now clusters of customers have been generated using the GA in the first phase of the solution method. In the second phase routes are constructed for the clusters and costs evaluated. The modified space-time nearest neighbour heuristic (Baugh routing-heuristic) described by Baugh et al. is used, in an extended version, for developing routes and calculating cost for each route. The Baugh routing-heuristic is described in section 4.3.

In the Baugh routing-heuristic the cost of a route is set to be the weighted sum of travel time and time window violations. The hard constraints that are included into the heuristic are the routing, precedence and capacity constraints. The depot, maximum route duration and maximum ride time constraints are not included. Neither are excess ride times, waiting times with passengers in the vehicle nor route duration. The missing constraints and cost factors are added to the heuristic. Service times at each location are also added to the heuristic.

The method of choosing the first customer in a route is also altered. In the Baugh routing-heuristic the first customer is chosen to be the customer with the earliest pickup time, which does not give good results for the data used here, since part of the pickup locations have no time window associated to them. It is the case for outbound customers. This means that the time window is set to the whole planning horizon ($[0, T]$) and the lower time window is zero. Therefore these customers are often chosen as first customers, the drop off time had no influence. In order to account for this, the first customer to visit is instead chosen to be the customer with the earliest latest pick up time. For those customers that have no pickup time windows there are assigned latest pickup times based

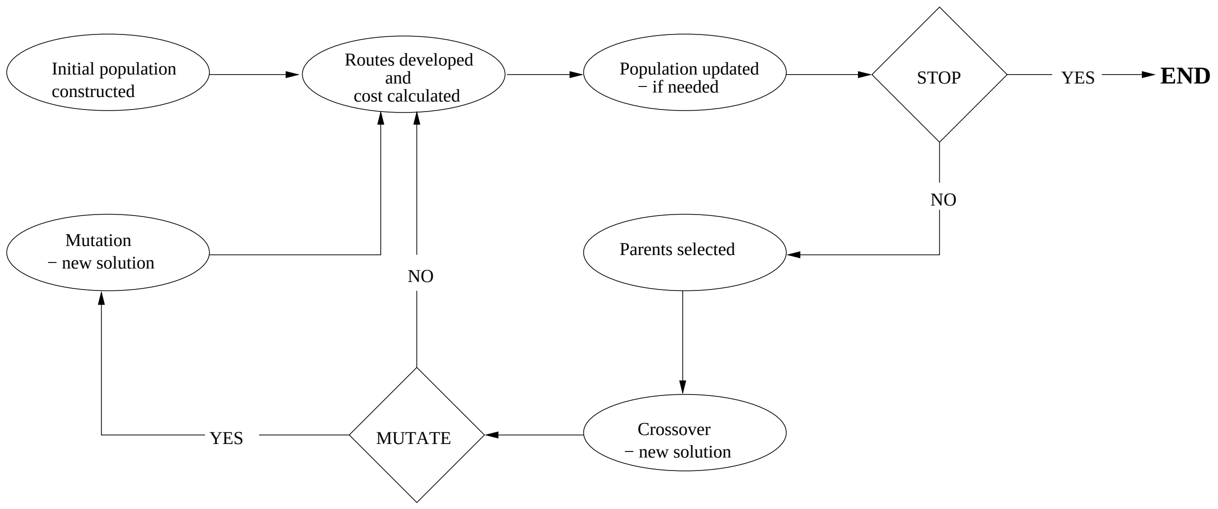
Initial population constructed
Routes developed and cost calculated
Population updated − if needed
STOP
YES
END
NO
Mutation − new solution
NO
Parents selected
YES
MUTATE
Crossover − new solution

Figure 5.6: An overview of the initial heuristic

on the drop off time windows. The latest pickup times for these customers are the upper time limit for the drop off location subtracted by direct transportation time and service time at the pickup location. An illustration of the latest pickup time calculations for an outbound customer is shown in figure 5.5.

If a vehicle arrives too early to a location it has the possibility of waiting. In the Baugh routing-heuristic this is included but the waiting times are not calculated specifically nor are they a part of the cost function. This is altered, to keep track of the waiting times, along with the total waiting times experienced by the customers. The total waiting time by customers is added in a weighted form into the cost function.

5.3.3 Overview of GA1

In this section the initial heuristic, GA1, will be summarized. Figure 5.6 gives an overview of the structure of the initial heuristic and algorithm 3 presents the pseudocode for GA1.

GA1 starts by constructing randomly an initial solution, where the customers are legally clustered, and then routes are developed for these clusters in the initial population and costs calculated. The cost of each individual is set to be the raw objective function value for each solution. Next the iterative process starts. A pair of parents are chosen, one randomly and the other by the roulette wheel method and they produce an offspring through a simple crossover. In the crossover one new cluster is constructed by mixing two clusters of the parents. The other clusters are the same as the roulette wheel chosen parent. With a probability P_μ the offspring is mutated, using a simple mutation that reallocates one randomly chosen customer to a new randomly chosen cluster. Routes are constructed for the new solution and costs evaluated. The population is updated by choosing a random solution from the bad part of the current population. This is a version of the steady-state/incremental genetic algorithm. The iterative process stops after a certain number of iterations.

5.4 Improvements to GA1

The possibilities of improving the initial heuristic are countless. In this section the improvements that will be tested in the next chapter are described.

Number of offsprings in each iterations set to M - GA2

Some experimental results presented in the literature [14] indicate that it is a better idea to construct generations that are of the same size as the population size, instead of just 1 as is the case in GA1. Therefore this will be the first effort on improving GA1. When updating the population after each iteration the new population will be the best of current population and the new generation with some random members to increase diversity.

Number of duplicates decreased - GA3

Duplicates distort the selective process and waste computational resources. They are therefore to be avoided if possible. In GA3 the probability of duplicate chromosomes is reduced by mutating the offsprings that have the same fitness values as their parent 1 and therefore represent the same solution. That will produce a different solution to the problem where one customer has been reallocated. This procedure will not check whether or not the solution presented by the mutated offspring is already present in the population. The reason for that is to save computational time.

Randomness reduced - GA4

In this improvement different methods for reducing the randomness of GA1 is described.

i. *Parent 1 always better than parent 2.* The parents are chosen in the same manner as in GA1, i.e. parent 1 is chosen using the roulette wheel method and parent 2 is chosen randomly. If this selection results in parent 1 to have a higher fitness value than parent 2 the parents are switched. Parent 1 is now parent 2 and parent 2 is now parent 1. The offspring always inherits most of its genes from parent 1 and the offspring will therefore always inherit more from the better parent in this improvement.

ii. *The template not constructed completely randomly.* The template used in the crossover in GA1 is constructed randomly. The template indicates which genes are to be taken from which parent. In this improvement the construction of the template is modified. The probability of taking genes from parent 1 is increased.

iii. *Select parent 1 from the best individuals in the population for the last part of the iterations.* For fine tuning the solution, i.e. finding a local optimum, the selection of parent 1 in the last iterations is altered in this improvement. The parent 1 is selected using the roulette wheel method for the first iterations as before but in the last iterations parent 1 is selected randomly among the best individuals (with the lowest fitness values). In the last part of the iterations it is the relative order of individuals as that is the basis for selection of parent 1 instead of the raw objective value as is used in the roulette wheel method.

Heuristic run twice while keeping the total number of iterations constant - GA5

The iterations have been set to a fixed number. It is possible to run the heuristic several times, each time using only part of the iterations, while the total number of iterations is kept constant. A variation of running the heuristic several times without changing the total number of iterations is tried in GA5. In GA5 the heuristic is run normally but after a certain number of iterations the current population is mutated (new mutation operator) and the mutated population is used as a new initial solution for running the heuristic again. The new mutation aims at getting to another part of the solution space. This is performed by changing all the 0s to 1s and vice versa in the chromosomes. Thereafter the chromosomes are corrected so that they represent legal solutions. After a certain number of iterations the last populations from each run are combined. The half of the best individuals from each population is used to construct a new population. When selecting the best individuals from both populations it is necessary to order the individuals and that is performed using Bubble sort[1]. The iterations in GA5 are finished normally using the newly combined population.

[1]Bubble sort is a simple way sort an array of objects but not the fastest. The basic idea is to compare two neighboring objects, and to swap them if they are in the wrong order. In the first iteration all objects are checked but in each of the following iterations the number of objects checked is reduced by one.

Algorithm 3 GA1

```
' Read data
 1: $M \leftarrow$ GetPopulationSize
 2: $G \leftarrow$ GetNumberOfGenerations (iterations)
 3: $P_\mu \leftarrow$ GetMutationProbabilty
 4: $Z \leftarrow$ GetLevelOfWorstIndividuals
 5: $car \leftarrow$ GetNumberOfAvailableVehicles
 6: $stops \leftarrow$ GetNumberOfStops (2×number of customers)
    ' Initial population constructed
 7: $\mathcal{P} \leftarrow$ RandomPopulationConstructor($M, car, stops$)
    ' Routes generated for initial population and costs calculated
 8: for ($i = 1 : M$) do
 9:    for ($j = 1 : car$) do
10:       $cluster \leftarrow$ GetCluster($\mathcal{P}, i, j$)
11:       $route \leftarrow$ MakeRoute($cluster$)
12:       $routecost \leftarrow$ GetRouteCost($route$)
13:    end for
14:    $cost(i) \leftarrow \sum_j routecost$
15: end for
    ' Genetic algorithm
16: for ($i = 1 : G$) do
17:    $random \leftarrow$ GetRandomInteger
       ' Select Parent 1 using the roulette wheel method
18:    for ($j = 1 : M$) do
19:       $\frac{1}{R} \leftarrow \sum_j \frac{1}{cost(j)}$
20:    end for
21:    for ($j = 2 : M$) do
22:       $P(j) = P(j-1) + \frac{R}{cost(j)}$
23:    end for
24:    for ($j = 2 : M$) do
25:       if($P(j-1) < random < P(j)$) $Parent1 = j$
26:    end for
       ' Select Parent 2 randomly
27:    $Parent2 \leftarrow$ GetRandomInteger
       ' One offspring created using crossover
28:    $Offspring \leftarrow$ Crossover($Parent1, Parent2$)
       'With probability $P_\mu$ apply mutation
29:    $Offspring \leftarrow$ Mutation($Offspring$)
       ' Routes for offspring generated and costs calculated
30:    for ($j = 1 : car$) do
31:       $cluster \leftarrow$ GetCluster($Offspring$)
32:       $route \leftarrow$ MakeRoute($cluster$)
33:       $routecost \leftarrow$ GetRouteCost($route$)
34:    end for
       ' Update population, a random solution of the $M/Z$ worst chosen
35:    $Out \leftarrow$ Select($cost, M/Z$)
36:    $\mathcal{P} \leftarrow \mathcal{P} + Offspring - Out$
37:    $cost \leftarrow$ Update
38: end for
39: Return best solution in $\mathcal{P}$
```

Chapter 6

Experimental results

This chapter presents the experimental results obtained from using the chosen solution approach for solving the formulated DARP. The solution approach is based on the cluster-first, route-second strategy. The genetic algorithm is used in clustering while the Baugh routing-heuristic is used to generate routes for the clusters. The chapter begins by a description of the data that will be used in the experiments. Next the experimental results will be presented. The presentation of the experimental results will be divided into the following three main parts.

In the first part the main focus will be on testing and tuning the initial heuristic. First values for four parameters in the genetic algorithm are selected by performing tests on three data sets and selecting the combination that gives the best results. The parameters are population size, number of iterations, mutation probability, and the proportion of the population that can be replaced by a new solution. Secondly, experiments with the weights in the objective function are performed. It will be shown how they can be adjusted to take into account different points of a view about cost vs. level of service. The customer wants as high service as possible while the transportation operator wants to provide the minimal accepted service at the lowest cost. The results from these experiments and considerations will be used to select values of the weights and the parameters in the GA that will be used in tests performed later in this chapter.

In the second part improvements to the initial heuristic proposed in the last chapter will be tested. The effect of having the generation size equal to the population size, reducing the number of duplicate solutions present in each population, reducing the randomness in the heuristic and different iteration approaches will be examined. When each improvement has been examined it is either accepted or rejected, that is, either the improvement is used further or eliminated. The later improvements can therefore possibly include previous improvements as well as the improvement itself.

The third part of this chapter is devoted to giving a short comparison of the performance of the best heuristic obtained in this project and of the simulated annealing solution approach proposed by Cordeau and Laporte [4] for solving the dial-a-ride problem. They generated the random data sets that are used for testing and tuning the initial and improved heuristics in this chapter.

6.1 Data

To my knowledge there are no well-known and established benchmarks available in the literature of the version of DARP used in this project. The behaviour of the solution method proposed here can therefore not be tested using data instances that have been widely tested and the results cannot be compared to the optimum or the best results obtained previously.

The test instances that will be used for testing the solution method proposed are obtained from Cordeau and Laporte [4]. They created the test instances to analyze the behavior of the simulated annealing when solving the DARP. Cordeau and Laporte generated a set of 20 randomly generated test instances (data sets) according to realistic assumptions. The information regarding time window widths, vehicle capacity, route duration, and maximum ride time is provided by the Montreal Transit Commission (MTC).

In the test instances there are between 24 and 144 customers. The first half of customers is assumed to consists of outbound customers while the remainder of the customers is assumed to be inbound. For each instance, the origin and destination locations are generated using a procedure that creates clusters of vertices around a certain number of seed points. All instances contain of a single depot and the location of the depot is set at the average location of the seed points. For a more detailed description of this procedure refer to Cordeau et al. [3].

For each instance the service time (s_i) in each location i ($i \in N$) is equal to 10 and the load change, l_i, in each location i is either 1 or -1. 1 for the pickup locations and -1 for the drop off locations, i.e. no customer has a companion travelling with them and all the customers need one seat. The depot location on the other hand has a service time and load change equal to zero, since no customers are entering or leaving the vehicle at the depots. Maximum route duration, r^k, in all instances is equal to 480, vehicle capacity, C^k, equal to 6 and maximum ride time, u_i, is equal to 90.

A time window $[a_i, b_i]$ is generated for each location. As mentioned in section 2.5 origin point of an inbound customer and destination point of an outbound customer are subject to time windows, while the other points have no customer specific time windows associated to them. The time windows for these points is therefore set to $[0, T]$, where T is the planing horizon. In the experiments it is equal to 1440, i.e. the number of minutes in one day - $24 \times 60 = 1440$. The time unit used is minutes.

Two groups of customer specific time windows are constructed in the data sets. The first one has narrow time windows while the second one has wide time windows. The time windows in the first group are constructed by first choosing an uniform random number,

Table 6.1: Size of data instances used in tests

	Customers	Vehicles
R1a & R1b	24	3
R2a & R2b	48	5
R3a	72	7
R4b	96	9
R5a & R5b	120	11
R6a & R6b	144	13
R07a & R07b	36	4
R9a & R9b	108	8
R10a & R10b	144	10

a_i, in the interval $[60, 480]$ and then choosing another uniform random number, b_i, in the interval $[a_i + 15, a_i + 45]$. For the wide time windows group a_i is chosen in the same manner but b_i is chosen in the interval $[a_i + 30, a_i + 90]$. Resulting in time windows $[a_i, b_i]$ $(i \in N)$. The test instances R1a to R10a have narrow time windows while test instances R1b to R10b have wide time windows.

Test instances R1a to R6a and R1b to R6b are generated in such a manner that the number of available vehicles in comparison to the number of customers is higher than in test instances R7a to R10a and R7b to R10b, e.g. in R6a has 13 available vehicles while R10a has 10 available vehicles, both instances having 144 customers.

All the test instances constructed by Cordeau and Laporte are available from the Internet at http://www.hec.ca/chairedistributique/data/darp/.

Table 6.1 gives the number of customers and available vehicles in the test instances that will be used in this project. Both test instances that have a large and small number of vehicles available proportional to the number of customers are chosen. In the solution procedure used in this report it is not possible to reject customers. The influence of the no reject procedure can therefore be investigated. As can be seen, the number of customers is between 24 and 144 and both the instances with narrow and wide time windows are chosen. The reason for not choosing all the data sets given by Cordeau and Laporte is because they do not report complete solutions to all the problems. All the problems in table 6.1 have a given solution except R4b and R7a, which will be used in parameter tuning.

The distance between any two locations i and j is set to be the Euclidean distance between the coordinates of locations i and j, $i, j \in A$. The speed of the vehicles is set to 1, so the transportation time $t_{i,j}$ is equal to the Euclidean distance between i and j. So the first term in the objective function 5.1 equals now the total weighted transportation distance.

6.2 Testing and tuning the initial heuristic - GA1

In this section the results from solving the DARP using the initial heuristic, GA1, are presented. GA1 is tested using three different values for four parameters in the genetic algorithm. The parameters found to give the best results in view of cost are chosen and used for the remainder of the tests performed in this chapter unless otherwise stated.

The solutions obtained by the heuristic vary in different runs of the same data set. It is therefore necessary to perform several runs for each data set. In this project 5 runs for each solution method on each data set will be performed and the average and best results reported. It is desirable to perform a higher number of runs than 5 but that is very time consuming and therefore 5 runs are chosen in this project.

6.2.1 Selecting values for the parameters in the GA

The values of four parameters in the genetic algorithm used in clustering the customers have yet to be declared. Those parameters are population size, M, number of generations (iterations), G, mutation probability, P_μ, and the proportion of the current population that can be replaced, Z. After performing some initial experiments the following values for the four parameters are chosen to be tested further:

$G = \{10.000; 15.000; 30.000\}$
$M = \{30; 50; 100\}$
$P_\mu = \{0,01; 0,05; 0,10\}$
$Z = \{0,10; 0,25; 0,50\}$

All 81 combinations of these parameter values are run on three data sets, R4b, R6a, and R7a. These data sets are chosen because they represent both large and small number of customers, narrow and wide time windows and tight and loose constraints on the number of vehicles compared to number of customers. Furthermore data sets R4b and R7a are also chosen because Cordeau and Laporte do not give a complete solution to these problems. They can therefore not be used in the comparison of different heuristics later in the chapter anyway.

The choice for the parameter values that are tuned is only based on the cost. Therefore it is the best and average costs obtained from running GA1 for all the combinations of the parameters for the three data sets are presented in tables 6.2, 6.3 and 6.4. The lines in the tables can be divided into three parts, one for each number of iterations (G) tried. Each of the three parts of lines can again be divided into three parts, one for each population size (M) tested. Lastly these three parts can be divided into three lines, one for each mutation probability (P_μ) tested. Three categories of columns are present in the tables, one for each part of the population that can be termed “bad” (Z).

Table 6.2: Average and best costs obtained for R4b in the parameter tuning. The bold highlights the best average cost obtained for 15.000 iterations and is used to scale the cost.

R4b			Z=0,50		Z=0,25		Z=0,10	
			avg	best	avg	best	avg	best
G=10000	M=30	Pμ=0,01	33236	26785	35051	32829	35958	30285
		Pμ=0,05	38114	31420	33733	30670	32119	27430
		Pμ=0,10	35195	33099	36508	32902	32367	27525
	M=50	Pμ=0,01	39706	32584	39627	37139	34920	27418
		Pμ=0,05	33826	25820	36115	32676	39462	33766
		Pμ=0,10	41973	40211	35408	29745	37189	32000
	M=100	Pμ=0,01	44469	37534	46359	39620	45097	41989
		Pμ=0,05	43063	40426	44659	41246	40690	35198
		Pμ=0,10	44820	43027	43098	37665	44059	42145
G=15000	M=30	Pμ=0,01	34076	27545	32153	25190	31822	27030
		Pμ=0,05	32201	27805	30985	27003	30704	27152
		Pμ=0,10	33817	31110	33078	27261	30052	26961
	M=50	Pμ=0,01	**28858**	25083	32615	27921	29858	23116
		Pμ=0,05	32786	28120	32531	31053	33411	28220
		Pμ=0,10	30644	25234	32546	30296	33250	29136
	M=100	Pμ=0,01	35429	30431	38846	35066	35509	31000
		Pμ=0,05	37653	34288	40020	34335	38816	36385
		Pμ=0,10	39534	32221	38402	32780	39993	32437
G=30000	M=30	Pμ=0,01	26522	23400	27409	25918	26667	22493
		Pμ=0,05	27062	24110	27838	24289	27089	22696
		Pμ=0,10	25039	19883	26653	25017	24542	19578
	M=50	Pμ=0,01	25529	20720	28134	20713	30544	25263
		Pμ=0,05	27100	25326	27043	25355	28141	24821
		Pμ=0,10	28634	24436	26108	22924	27031	22701
	M=100	Pμ=0,01	26940	22732	28873	28146	30292	26181
		Pμ=0,05	29628	26273	28793	23215	32917	26746
		Pμ=0,10	26612	19476	29398	24459	32068	30174

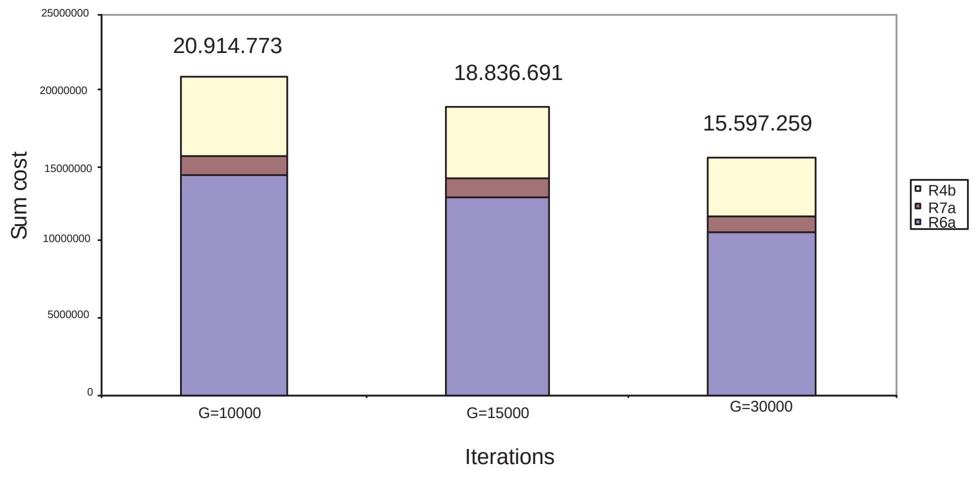

Figure 6.1: The total cost for all three data sets for the three number of iterations. The cost decreases with number of iterations.

Table 6.3: Average and best costs obtained for R6a in the parameter tuning. The bold highlights the best average cost obtained for 15.000 iterations and is used to scale the cost.

R6a			Z=0,50		Z=0,25		Z=0,10	
			avg	best	avg	best	avg	best
G=10000	M=30	Pμ=0,01	94754	86619	86130	80677	88731	76421
		Pμ=0,05	95265	77042	96126	88447	98344	92155
		Pμ=0,10	96287	91914	89039	72580	92779	83346
	M=50	Pμ=0,01	102551	89675	102043	84958	98156	87013
		Pμ=0,05	101775	91209	100271	92151	104844	92408
		Pμ=0,10	103225	93660	106121	97621	104782	97257
	M=100	Pμ=0,01	123050	109879	113652	98036	119761	108304
		Pμ=0,05	121809	113784	113990	91064	121300	115203
		Pμ=0,10	120890	106611	118530	102408	109100	95476
G=15000	M=30	Pμ=0,01	89123	79594	**83945**	78462	89357	73131
		Pμ=0,05	87086	77743	91769	87825	86882	66397
		Pμ=0,10	99905	84919	87491	77448	85391	71198
	M=50	Pμ=0,01	97617	94904	100878	88853	88719	74791
		Pμ=0,05	95963	91190	91781	83282	88759	77457
		Pμ=0,10	98306	85578	91254	80268	89670	81995
	M=100	Pμ=0,01	101332	79328	102886	94706	103477	91961
		Pμ=0,05	106234	100581	105710	103504	100841	95297
		Pμ=0,10	96084	86183	113114	104185	107298	92054
G=30000	M=30	Pμ=0,01	80850	77854	70704	66274	74336	63040
		Pμ=0,05	85234	76001	75796	64900	76224	66567
		Pμ=0,10	74222	61250	70174	63718	73249	65153
	M=50	Pμ=0,01	77591	65891	77815	70128	74957	68186
		Pμ=0,05	75899	64319	76668	67720	75976	63897
		Pμ=0,10	72265	60873	73377	60912	75427	67690
	M=100	Pμ=0,01	85162	75922	87715	80421	85181	68275
		Pμ=0,05	85081	72969	84251	74251	88205	85478
		Pμ=0,10	81092	68230	82603	71086	79743	74837

In tables 6.2, 6.3 and 6.4 it can be seen that the cost decreases as the number of iterations increases. Figure 6.1 shows that the total cost for all runs of 15.000 iterations is 10% lower than for 10.000 iterations. The total cost for 30.000 iterations is the lowest, or 17% lower than for 15.000 iterations. It seems therefore to be wise to set the number of iterations to 30.000 or higher. The drawback is the computational time, which seems to be linearly dependent on the number of iterations. The CPU time for R6a, the largest data set, is about 150 minutes for 30.000 iterations, 75 minutes for 15.000 iterations and 50 minutes for 10.000 iterations. In order to be able to complete the testing of GA1 and the improvements in a reasonable amount of time as well as obtaining good solutions to the problem I decided to use 15.000 iterations for further studying and testing. The remarks and conclusions in this chapter are based on the results obtained using 15.000 iterations.

General pattern about the other three parameters is hard to detect. When the problem size is small a large population results in the lowest average cost, but when the size of

Table 6.4: Average and best costs obtained for R7a in the parameter tuning. The bold highlights the best average cost obtained for 15.000 iterations and is used to scale the cost.

R7a			Z=0,50		Z=0,25		Z=0,10	
			avg	best	avg	best	avg	best
G=10000	M=30	Pμ=0,01	9548	8281	8658	7607	9613	8370
		Pμ=0,05	8942	8467	10297	8499	8150	7471
		Pμ=0,10	9184	7295	8201	6826	8818	6706
	M=50	Pμ=0,01	8646	7510	8505	7546	9122	8112
		Pμ=0,05	7916	6511	8035	6487	9433	8270
		Pμ=0,10	7513	6244	9003	7445	9192	7447
	M=100	Pμ=0,01	8762	8002	8941	7191	8762	7229
		Pμ=0,05	10600	8940	8667	7989	10020	9182
		Pμ=0,10	9660	8614	8518	6993	9114	7707
G=15000	M=30	Pμ=0,01	7810	6859	9004	6623	8702	7065
		Pμ=0,05	9194	7211	7985	7537	9268	7542
		Pμ=0,10	8765	7921	9319	6883	9155	7064
	M=50	Pμ=0,01	9574	8077	7776	7040	8645	7114
		Pμ=0,05	8253	7720	8497	6917	8401	7473
		Pμ=0,10	8976	7833	9407	7981	8664	6523
	M=100	Pμ=0,01	8724	7695	8670	6833	**7586**	6943
		Pμ=0,05	7862	7174	8234	7658	8261	7389
		Pμ=0,10	8580	5876	8339	7138	8940	7652
G=30000	M=30	Pμ=0,01	8255	7366	8068	7035	6928	6349
		Pμ=0,05	7570	6855	8437	7536	8017	6094
		Pμ=0,10	7256	6413	7845	6382	7717	6398
	M=50	Pμ=0,01	7361	6621	8005	6780	6978	6230
		Pμ=0,05	7951	6594	7977	6900	8361	7736
		Pμ=0,10	7860	7029	8158	7404	8403	7462
	M=100	Pμ=0,01	8109	5820	7512	6840	7418	6385
		Pμ=0,05	7888	6525	7455	6667	7689	7424
		Pμ=0,10	8015	7067	7837	6172	7632	7316

the problem is large it is gives better results to use a small population. One reason for this behaviour is that the total number of possible different individuals increases with the number of customers. When the size of the population is large the solution space is better covered but it takes a larger number of iterations to reach a local optimum or a near optimum solution. In order to perform a local search in the neighbourhood of a good solution that solution has to be used to produce new solutions in crossover. In the crossover one parent is chosen randomly and the second parent is chosen proportional to the "goodness" of the solution. When the number of individuals increases the probability of choosing each individual decreases and that goes also for the good solutions. Therefore the number of times the best individuals are used in crossover is lower than for small population. For 15.000 iterations $M = 30$ is best for R6a, $M = 50$ is best for R4b and $M = 100$ is best for the smallest data set R7a. That is, the 15.000 iterations are not enough for the larger data sets and as a result the smaller population sizes work better for them.

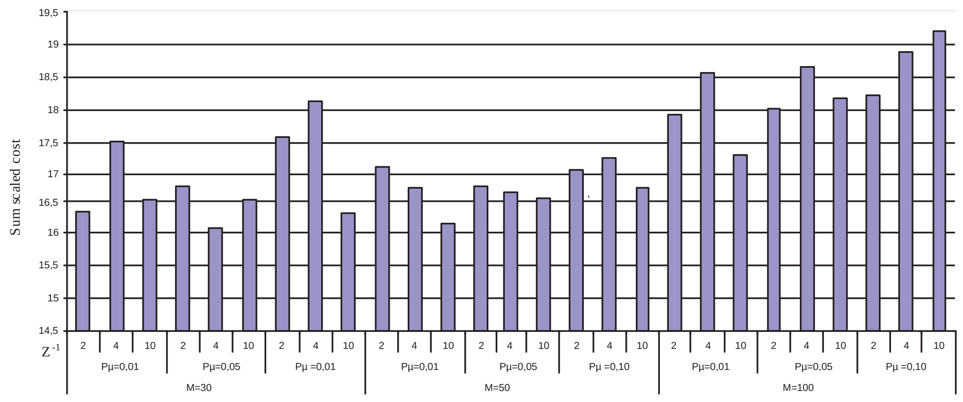

Figure 6.2: The total scaled cost for all combinations of M, P_μ and Z when $G = 15.000$ for R4b, R6a and R7a.

The overall best results for the mutation probability is $P_\mu = 0,05$ closely followed by $P_\mu = 0,01$. For the size of the proportion of the population that can be subject to replacement $Z = 0,10$, i.e. it is possible to choose between $\frac{M}{10}$ individuals to be replaced, gives the overall best results. These results are obtained by summing the cost from all the runs for all the combinations of three parameters while keeping P_μ or Z constants.

The data sets have from 36 to 144 customers and the average total cost ranges from 7.586 to 123.050. Thus, a raw average will not be a good choice to select the best combination of the three parameter values. If the raw average would be used then the data sets with higher cost will have greater weight than the data sets with smaller costs. See for example in figure 6.1 that the total cost for R6a is much larger than the total cost for the other two smaller data sets. Therefore R6a is a dominant factor in the average total cost for all the runs. For this reason the costs are scaled by dividing the costs of each combination in each data set by the minimum average cost for that data set. This minimum average is indicated by bold in tables 6.2, 6.3 and 6.4. The result of scaling the cost like this is that the maximum range for the scaled cost in the three data sets is [0,72 ; 1,67] and the range is similar for all three data sets. Therefore the data sets with higher cost do no longer weigh more than the less expensive ones. A summary of the scaled costs for 15.000 iterations is given in figure 6.2. In the figure, each column represents the sum of the scaled costs of the results for data sets R4b, R6a and R7b for each combination of the three remaining parameters. As we can see in the figure two combinations stand out as the best.

The two best combinations are:

$$M = 30 \quad P_\mu = 0,05 \quad Z = 0,25$$

and

$$M = 50 \quad P_\mu = 0,01 \quad Z = 0,10$$

with total scaled costs 16,10 and 16,15 respectively. I decided to use the latter combination since the overall scaled cost for the three data sets is the lowest for population size of 50.

6.3 Selecting values for the routing parameters in GA1

The routing parameters are the weights, w_1 to w_8, in the objective function, see section 5.2. The relative values of these weights in comparison with one another give the cost factors in the objective function different importance levels. The higher the value of the weight the more important the cost factor the weight is multiplying becomes. The size of the cost factor itself also plays a role in how the weights are to be set. That is, small cost factors need greater weights than larger cost factors in order to have a bearing in how the solutions will develop. The problem of the different sizes of the costs can be solved by scaling the costs before they are entered into the fitness function but that will not be performed here. Instead the weights will be set to take the different sizes of cost factors into consideration. The weights are:

- w_1: weight on driving time, which equals distance here since speed is equal to 1.
- $w_{3'}$: weight on excess passenger ride time.
- w_4: weight on passenger waiting time.
- w_5: weight on route duration.
- w_6: weight on time window violation.
- w_7: weight on passenger ride time violation.
- w_8: weight on route time violation.

Note that w_2, weight on number of vehicles used, has been omitted from the model since we have a constant number of vehicles available, see section 5.2.

In the parameter tuning of the GA in the last section the values for the weights are set to:

$$w_1 = 8, \quad w_{3'} = 3, \quad w_4 = 1, \quad w_5 = 1, \quad w_6 = 20, \quad w_7 = 20, \quad w_8 = 20$$

When selecting the weights no one set of weights represents the "right" set giving the best results for some data instances, as is for the parameters in the GA1. The selection of weights is performed by people and they can represent different points of a view. In this section two sets of weights will be tested on all the data sets presented in table 6.1 except the three that have been used in the tuning of the GA parameters. One set of weights is supposed to represent customers choice and the second the transportation operator choice.

The results from the five runs performed for the 13 data sets used in testing the initial and improved heuristics are summarized into tables. Each data instance is run 5 times and the tables present the average results from the data obtained in these runs along with the best total cost found for each instance. The results tables are made up of eleven columns:

- The first column gives the name of the data set for each line.
- The second and third columns give the average and best total cost.
- The fourth column presents the average passenger cost, i.e. average total cost divided by number of customers. The number of customers can be seen in table 6.1.
- In the fifth column total distance for the routes in a solution is reported. The routes equal the number of vehicles available and the number of vehicles available in each data set can be seen in table 6.1.

- The total route duration is presented in the sixth column.
- The seventh column gives the total ride time for the customers.
- The eighth column presents the waiting time experienced by the customers.
- The ninth column gives the solutions average load of customers in the transportation vehicles. It is calculated based on the following equation 6.1:

$$\text{Avg load} = \sum_{p=1}^{m} \left(\frac{1}{N_p} \sum_{i=1}^{N_p} L_i\right) \tag{6.1}$$

  where N_p is the number of customer stops in route p, L_i is the load after servicing at customer stop i and m is the number of vehicles used in the solution.
- The tenth column gives the total time window violation for all the stops in the routes of each data set.
- The eleventh column presents the CPU times in minutes. The CPU times are measured on an Intel Celeron CPU 2 GHz computer.

The reason for presenting these results for the solutions cost is to being able to investigate the development of the total cost, which is being minimized in the heuristic. It is also important to be able to study the route specific factors, distance, route duration, ride time, waiting time, load and time window violation. These factors represent the "real" cost experienced by the customers and/or the transportation operator. Another reason is being better able to compare the results for different heuristics.

6.3.1 GA1 - Customers choice

The customers in this section emphasize the level of service provided by the transportation operator. They want the bus to arrive on time and the ride time to be minimal. On the other hand they do not want the service to be very expensive and therefore they set their demands on service to reasonably high levels in their opinion. After performing some preliminary testing the resulting weights are:

$$w_1 = 8, \quad w_{3'} = 3, \quad w_4 = 1, \quad w_5 = 1, \quad w_6 = n, \quad w_7 = n, \quad w_8 = n$$

The weights on violating the relaxed constraints presented in the objective function (equation 5.1) is set to n, i.e. the total number of customers in each data instance. The reason for this choice is that in GA parameter tuning it is experienced that the violation increases with the number of customers, because the values for the cost factors for distance, route duration and ride time increase proportionally with the number of customers. The value of the cost terms for the relaxed constraints is not as dependent on the number of customers as the above mentioned factors. Therefore, it is decided to set the weights for the violation terms to n. The weight on total distance driven by the transportation vehicles is set to 8, because the customers are concerned with the transportation cost. The weight on excess ride time is set to 3, as the customers perceive that it is important that the transportation time is short. The weights on waiting time with customers is set to 1, because the customers generally do not mind very much waiting in the vehicle as long as the excess ride time is reasonable. The weight on route duration is also set to 1, because the size of the route duration is large compared to the other segments in the fitness function and the customers are not very concerned with the route duration. This set of weights is tested on the 13 data sets and the results can be seen in table 6.5.

Table 6.5: GA1 - Weights set by customers choice

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol.	[min]
R1a	5162	4729	215	316	1009	568	42	0,99	0	4,41
R2a	20893	14030	435	517	2015	1558	118	1,29	98	10,46
R3a	95377	88571	1325	1050	2844	3529	297	1,56	404	20,03
R5a	266671	237535	2222	1349	4419	5665	552	1,64	1027	55,74
R9a	433888	395670	4017	1379	3676	6267	250	2,01	2282	39,95
R10a	878412	769719	6100	1825	5085	8853	602	2,06	3597	63,41
R1b	4686	4165	195	293	995	540	17	0,94	3	4,41
R2b	17368	11626	362	557	1715	1472	57	1,22	22	10,47
R5b	138511	111773	1154	1353	4214	5377	306	1,64	258	55,82
R6b	262010	219236	1820	1778	5196	6648	397	1,60	522	76,54
R7b	12507	8302	347	485	1270	1169	44	1,24	32	7,03
R9b	252041	224697	2334	1369	3745	5832	319	1,84	932	37,89
R10b	670927	633757	4659	1779	4788	8378	129	2,03	2510	63,27
Total	3058453	2723812	25186	14048	40971	55855	3130	20,06	11687	449,44

We can see in the table that the average passenger cost varies from 215 to 6100 and increases with number of customers. One reason for this is that the violation weights are set to n, i.e. to the number of customers, and the violation of these constraints therefore cost more for the large problems. The probability of getting positive values for the relaxed segments of the fitness function is also greater for the larger data sets. Both total distance and route duration increases almost linearly with the number of customers. The total passenger ride time is different for the different types of problems present. The ride time is higher when the time windows are narrow and when the vehicles have increased average load. The total passenger waiting time in the next column seems to grow with number of customers except for the R10b, which is very low. The average loads in the vehicles range from 0,94 to 2,06. The top utilization of the transportation vehicles is achieved when the number of vehicle compared to number of customers is low, as is the case for R9a, R10a, R7b, R9b and R10b. The CPU times increase with the number of customers and are higher for data sets with narrow time windows. The CPU times for the data sets with fewer number of vehicles per customers is lower than for the data sets with a higher number of vehicles. This is because the routing algorithm used in the heuristic uses about 99% of the CPU time and when the number of vehicles is increased the routing algorithm is called more often.

Figures 6.3, 6.4 and 6.5 show how the routes look like for one randomly chosen solution for data instance R1a. Figure 6.6 depicts all the three routes together. As we can see the routes are not localized, meaning that the customers are not divided into groups by location. The stronger factor seems to respect the time windows. It can also be seen that the customers very often are driven directly from their pickup to their drop off location (pickup locations have numbers ranging from 1 to 24 and the drop off locations have the

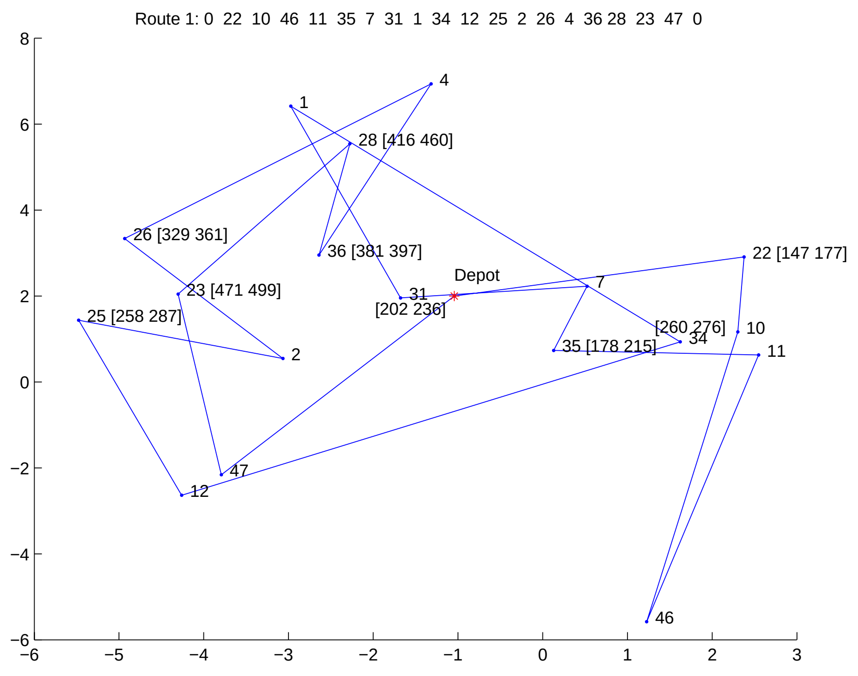

Figure 6.3: Route 1 for data set R1a (24 customers). The numbers 1 to 24 by a node indicate the pickup location of customers 1 to 24 and numbers 25 to 48 by a node indicate a drop off location for customers 1 to 24. The numbers in brackets [] are the time windows for the locations.

same number as the pickup location adding 24 and 0 indicates the depot). The level of customer service is therefore high as the customers want.

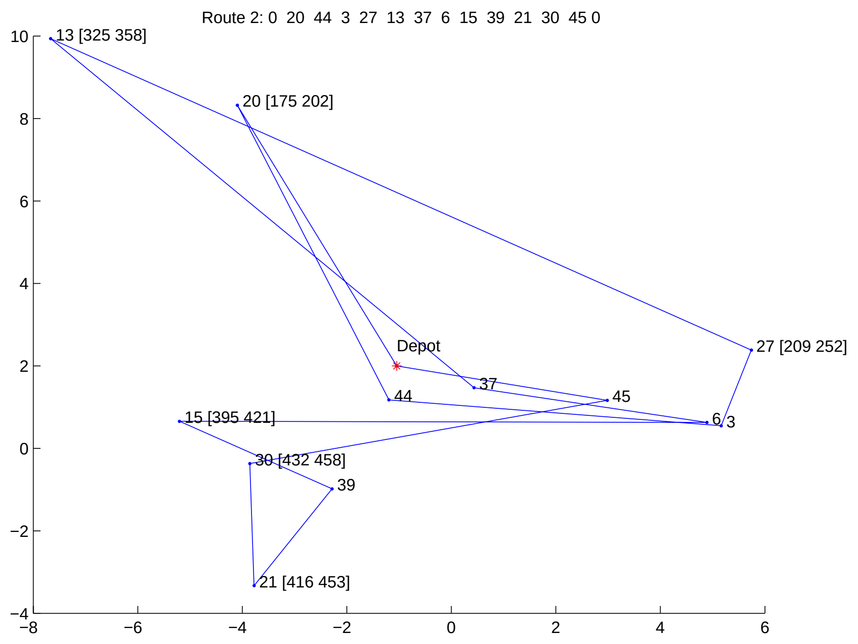

Figure 6.4: Route 2 for data set R1a (24 customers). The numbers 1 to 24 by a node indicate the pickup location of customers 1 to 24 and numbers 25 to 48 by a node indicate a drop off location for customers 25-24 to 48-24. The numbers in brackets [] are the time windows for the locations.

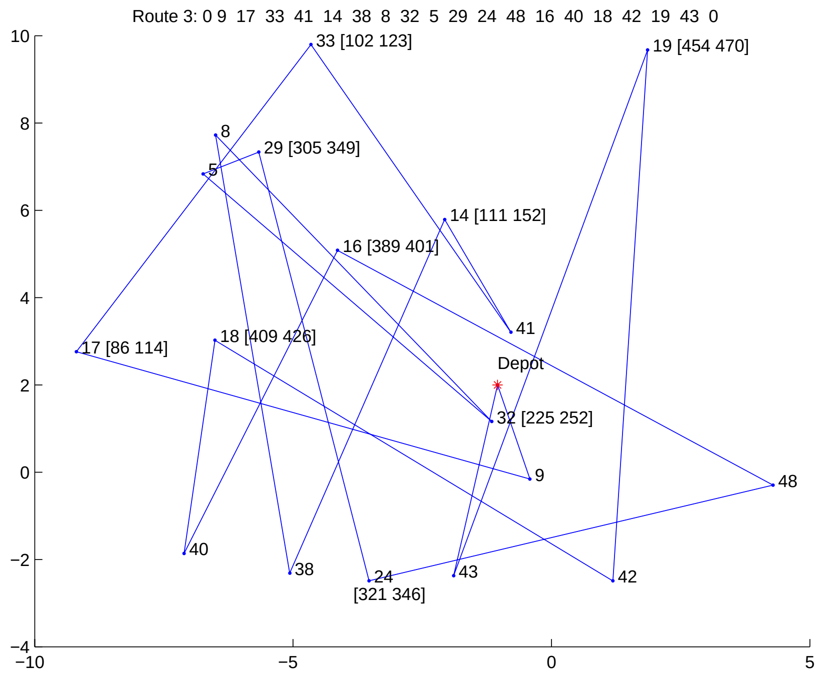

Figure 6.5: Route 3 for data set R1a (24 customers). The numbers 1 to 24 by a node indicate the pickup location of customers 1 to 24 and numbers 25 to 48 by a node indicate a drop off location for customers 25-24 to 48-24. The numbers in brackets [] are the time windows for the locations.

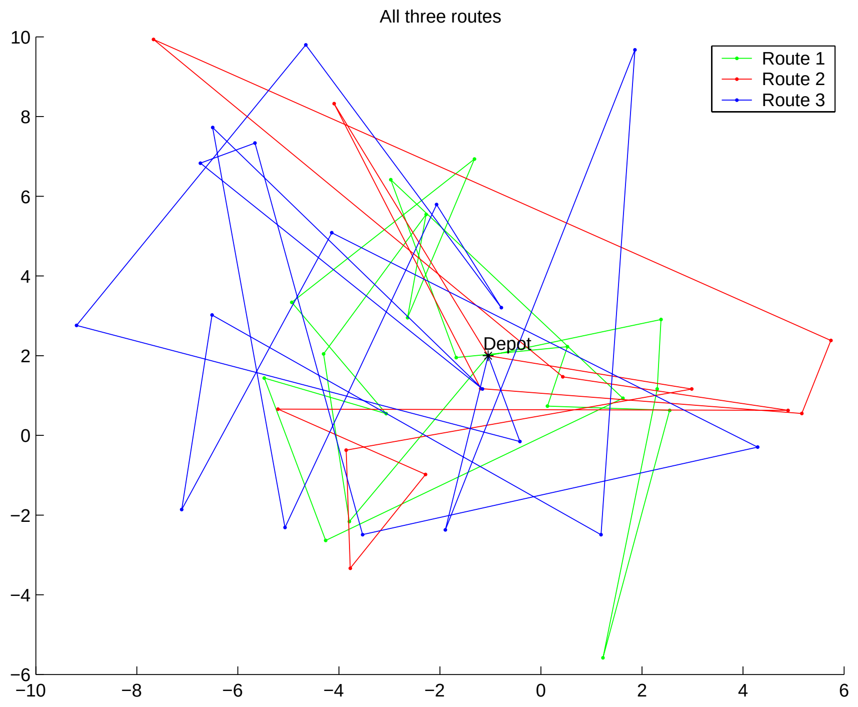

Figure 6.6: All the three routes for R1a together.

6.3.2 GA1 - Operators choice

The transportation operator has different wishes than the customer. The operator wants to provide minimal service that is still within the service requirements as inexpensively as possible. The operator will therefore set high weights on the violation terms as is also the case with the customers choice but he puts higher weight on route duration and no weight on the excess ride time. The weights set by the transportation operator are:

$$w_1 = 8, \quad w_{3'} = 0, \quad w_4 = 1, \quad w_5 = 2, \quad w_6 = n, \quad w_7 = n, \quad w_8 = n$$

Results from using this set of weights on the same 13 data instances as before can be seen in table 6.6. The main difference from table 6.5 is that distance, route duration and passenger waiting time have decreased while the average utility of the vehicles has increased. The down side is that customer ride time has increased as well as time window violation. From the operators point of a view it is satisfactory that the ride time increases as long as it does not become greater than the maximum ride time, because the operators concern is to hold the service requirements, e.g. the maximum ride time, and if the ride times of customers is below the maximum the operator is satisfied. How much the ride time is below the maximum is of no concern to the operator. The increased time window violation, from the results obtained using the customers choice of weights in table 6.5, on the other hand is not good. The increase in time window violation could be expected since the weight on route duration has been doubled. Larger weights would have to be placed on time window violation for the operators choice of weights in order to get better results for the time window violation.

Table 6.6: GA1 - Weights set by operators choice

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
R1a	4632	4183	193	283	910	986	57	1,52	0	4,44
R2a	26140	17487	545	521	1919	2146	128	1,66	149	10,44
R3a	79594	68017	1105	1018	2757	3683	200	1,69	386	20,31
R5a	317966	277191	2650	1309	4240	6977	504	2,02	1063	56,81
R9a	475031	409457	4398	1326	3588	7103	131	2	2502	43,24
R10a	908227	792889	6307	1781	4922	9762	249	2,34	3586	64,32
R1b	3552	3283	148	252	762	1015	7	1,65	0	4,46
R2b	16683	9648	348	525	1591	2053	92	1,60	95	10,74
R5b	168270	132152	1402	1317	4167	6025	344	1,79	477	56,34
R6b	313292	238888	2176	1734	5096	7569	354	1,83	705	77,28
R7b	9215	6171	256	454	1341	1606	112	1,57	4	7,02
R9b	279869	242116	2591	1307	3610	6490	151	2,17	974	42,94
R10b	880341	794942	6113	1737	4767	9441	212	2,31	3503	64,78
Total	3482812	2996424	28233	13563	39669	64857	2539	24,47	13444	463,11

6.3.3 Weights chosen and convergence

Since I can better relate to the customers points of a view than the operators I have chosen to use the customers choice of the weights in the tests that are presented later in this chapter.

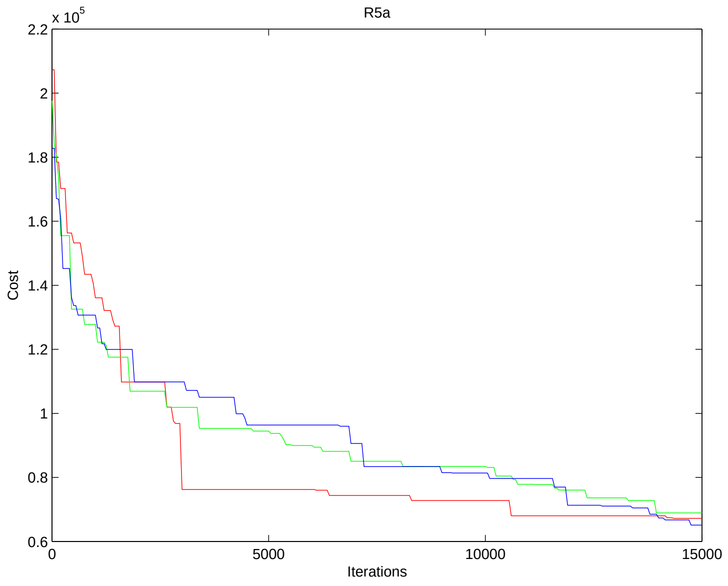

Figure 6.7: Graphs cost vs. number of iterations for 3 runs for R5a.

Figure 6.7 presents cost vs. number of iterations for three results obtained for R5a. As we can see the cost is high in the first iterations and then decreases very rapidly in the first part of the iterations (around the first 2000 iterations). The rate of cost decrease then diminishes as the number of iterations increases. The results for the three runs are similar.

6.4 Improving the initial heuristic

Now all the parameters in GA1 have been determined. It is time to test the improvement strategies described in chapter 5. First the size of each generation will be changed from 1 to M, then the number of duplicates is reduced, next the randomness of GA1 will be reduced and lastly the heuristic will be executed twice in a run. Each improvement is tested and either accepted or rejected. When an improvement is accepted it will be a part of later improvements, if rejected it will be eliminated from further testing. The aim is to find the best procedure for solving the formulated DARP. The criteria set for acceptance is that the total average cost decreases and for no more than four individual data instances the average passenger cost increases. The larger data sets dominate the average cost and in order to take better account for the smaller data sets the latter part of the acceptance criteria is set. Even though the selection criterion is purely based on the cost the results for the routing factors will also be presented. It is interesting to follow their development as well as the cost and the routing factors will be used later in the chapter for comparing the results obtained by the best procedure to the results obtained by Cordeau and Laporte [4]. The tables in this section have the same structure as is used in testing and tuning the initial heuristic, which is described in section 6.3.

6.4.1 Size of generation equals population size - GA2

The first improvement to be tested is denoted GA2. In GA2 a whole population is created in each iteration (M offsprings) and the M best individuals of the current population and the new generation chosen as the next population.

The CPU time is linearly dependent on the number of iterations and the most time consuming factor in each iteration is generating the routes for the customer clusters (about 99% of the CPU time). Now the number of clusters in each iteration is multiplied by M from what is in GA1. In order to get similar CPU times the number of iterations performed in this testing is divided by M, i.e. $G/M = 300$ iterations. The results for GA2 can be seen in table 6.7.

Table 6.7: GA2 - Whole population created in each iteration

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
R1a	7803	5941	325	325	1034	14120	89	1,24	17	4,48
R2a	131481	79614	2739	556	2098	53975	335	1,65	1082	10,31
R3a	300327	205989	4171	1051	3149	98504	781	1,75	2488	19,94
R5a	2078834	733497	17324	1345	4578	183399	1154	2,02	3996	61,34
R9a	958734	784024	8877	1406	3938	216099	633	2,14	6113	37,49
R10a	1788626	1716468	12421	1830	5400	283113	1000	2,06	8839	71,07
R1b	8501	7135	354	315	954	15273	52	1,36	34	4,81
R2b	73532	38665	1532	555	1790	53109	405	1,62	506	10,58
R5b	543125	474714	4526	1384	4577	142507	856	1,63	2464	63,76
R6b	935492	760456	6496	1763	5423	182607	922	1,67	4009	74,55
R7b	29026	24560	806	470	1324	30680	123	1,50	228	8,09
R9b	666381	551954	6170	1365	4002	194696	644	1,98	3692	37,90
R10b	1453372	1383913	10093	1809	5249	294247	834	2,18	6491	77,03
Total	8975233	6766929	75835	14175	43518	1762329	7828	22,79	39959	481,36

The main difference from the results obtained by GA1 (table 6.5) is that the cost has almost tripled and the ride time and time window violation have increased drastically. The other fact such as distance and route duration are similar. All in all improvement GA2 gives much worse results than GA1. The reason for this much worse results is that only 300 iterations are performed and the heuristic is not able to search the solution space nor converge towards a local optimum. However, the CPU time is similar as for GA1 and therefore it is not justifiable to increase the number of iterations. GA2 is rejected *rejected* based on the computational times.

6.4.2 Number of duplicates reduced - GA3

The next improvement aims at reducing the number of duplicates in the population. Instead of checking all the members of a population it simply checks if a new offspring is a duplicate of parent 1. If that is the case the offspring is mutated and thereby a different individual created. This should increase diversity of the members of the population and lead to a better investigation of the solution space. The results from GA3 can be seen in table 6.8.

Table 6.8: GA3 - Number of duplicates reduced

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
R1a	4749	4486	198	300	1038	530	30	0,98	1	5,09
R2a	19247	17222	401	529	1971	1675	113	1,38	71	11,54
R3a	74278	65244	1032	1064	2808	3387	248	1,55	248	23,68
R5a	286856	264825	2390	1362	4330	5899	540	1,69	1031	57,36
R9a	405899	363147	3758	1374	3656	6187	172	2,01	2166	39,15
R10a	876352	830424	6086	1794	5055	8940	489	2,10	3587	65,60
R1b	4711	4479	196	306	903	561	4	0,94	2	5,17
R2b	13132	8872	274	551	1651	1368	20	1,21	17	10,94
R5b	131227	112076	1094	1368	4193	5029	281	1,54	224	57,29
R6b	238654	184394	1657	1813	5101	6326	288	1,55	555	78,02
R7b	9081	8074	252	462	1257	1098	67	1,17	4	7,88
R9b	208874	169310	1934	1309	3664	5315	194	1,75	769	39,16
R10b	625965	544459	4347	1734	4813	7972	270	1,98	2271	64,60
Total	2899023	2577013	23619	13964	40441	54285	2715	19,86	10946	465,49

The results for GA3 are compared to the results from GA1 (table 6.5) and it is noticed that in all areas except vehicle utilization there is an improvement. The average passenger cost decreases for all data instances except for R1a and R1b, the smallest instances, where an increase of 2,5% and 32,4% respectively occurs. The average passenger cost for the larger data instances decreases up to 28,9% in R10b. The total average cost decreases and an increase in average passenger cost occurs in two data instances, therefore the method is *accepted*. Finally it is noticed that by using this method the number of duplicates in the final population is greatly reduced, e.g. for R5a the maximum number of duplicates for the last population is 36 using GA1 but 1 using GA3.

6.4.3 Reducing the randomness in the heuristic - GA4

Several parts of the solution process depends on randomness. It is therefore worth while checking if reducing the random behavior will improve the results. In this section three methods of reducing the use of random numbers will be tested. First, parent 1 is set to always represent the better parent, second, the template in the crossover procedure is modified and third, the selection of parent 1 in the last part of the iterations is limited to the best individuals in the population.

Parent1 always better than parent 2 - GA4i

In GA4i the parents are chosen in the same manner as in GA1 to GA3. But if parent 2 turns out to be the better than parent 1, i.e. has lower cost, the parent numbering is switched, so that parent 1 always represents a better individual than parent 2. This means that the offspring always inherits the major part of its genes from the parent with the better fitness value. The results for GA4i are shown in table 6.9.

Table 6.9: GA4i - Parent1 always better than parent 2

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
R1a	5037	4605	210	290	1019	580	41	1,00	5	6,21
R2a	19611	13815	409	523	1927	1536	91	1,32	104	11,74
R3a	64481	52726	896	1042	2812	3301	257	1,51	234	21,76
R5a	205320	190686	1711	1374	4353	5399	432	1,60	638	59,14
R9a	333973	286677	3092	1335	3579	5991	178	1,99	1622	40,69
R10a	741004	620435	5146	1784	4955	8380	440	2,02	2845	66,60
R1b	4680	4113	195	294	939	563	0	1,02	2	6,25
R2b	13969	8812	291	530	1627	1325	37	1,18	28	11,81
R5b	126472	103218	1054	1376	4325	4966	262	1,55	214	58,89
R6b	207969	147452	1444	1756	5179	6023	252	1,50	462	80,75
R7b	9204	7499	256	463	1250	1076	21	1,22	7	8,19
R9b	167828	126355	1554	1359	3725	5402	178	1,79	483	40,75
R10b	591027	485320	4104	1803	4885	8102	304	1,99	2061	72,65
Total	2490574	2051715	20361	13930	40574	52644	2493	19,71	8703	485,44

When comparing the results of GA4i with the results of GA3 (table 6.8) we can see that improvements have occurred in all categories except for route duration which has slightly increased. The total average cost decreased 14% but in four data instances (R1a, R1b, R2b and R7b) the average passenger cost increases. These four instances are the four smallest data instances tested and three of them have wider time windows. This improvement therefore lies on the border of being accepted, but it should be noted that in one (R1a) of the four the increase is almost insignificant (0,5%). The improvement, GA4i, is therefore *accepted*.

Template generated in a less random manner - GA4ii

The next improvement is to modify the template that is used in the crossover procedure. The template is used as a recipe in the crossover, i.e. it indicates which genes are to be taken from which parent. The template is generated in a complete random manner in GA1 to GA4i. In GA4ii the template is created by increasing the probability of selecting genes from parent 1. It is not wise to increase the probability very much, especially in the beginning of the iterations, therefore the probability is set to 60%. That is, the probability of choosing genes from parent 1 (the better parent) is 60%. The results for GA4ii are given in table 6.10.

Comparing the results from the 60% template and the results from GA4ii we can see that they are very similar in most categories. The total average cost has though decreased 6,4%. The average passenger cost is lower or similar for all data sets except R3a and R5a

Table 6.10: GA4ii - Template generated with 60% probability of choosing genes from parent 1

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
R1a	4696	4193	196	309	1041	477	29	0,89	1	5,57
R2a	19426	14003	405	539	1969	1367	81	1,18	98	11,43
R3a	65306	47801	907	1047	2779	3081	144	1,48	232	21,58
R5a	213420	180109	1778	1350	4250	5099	286	1,57	852	58,23
R9a	333283	299844	3086	1343	3597	6251	132	2,05	1488	40,78
R10a	740890	648300	5145	1811	5006	8413	401	2,00	2989	65,98
R1b	4762	4391	198	284	907	630	5	1,10	0	5,46
R2b	13580	10732	283	561	1719	1214	53	1,07	10	11,72
R5b	98111	74091	818	1344	4296	4615	221	1,45	119	58,93
R6b	185169	156823	1286	1799	5309	6134	361	1,53	384	81,23
R7b	9169	6682	255	478	1299	990	27	1,09	6	8,29
R9b	167709	155366	1553	1372	3679	5362	166	1,77	362	44,66
R10b	474758	419246	3297	1740	4733	7969	202	2,00	1548	66,41
Total	2330280	2021580	19206	13976	40584	51600	2109	19,18	8088	480,28

but in those cases it did not increase much, 1,2% and 3,9% respectively. The results are satisfactory and the improvement is *accepted*.

Parent 1 selected from the best individuals for the last iterations - GA4iii

The last effort in reducing the randomness of the genetic algorithm is to alter the selection mechanism of parent 1 for the last part of the iterations. Parent 1 is now chosen stochastically using the roulette wheel method. In GA4iii this is altered for the last 1000 iterations where parent 1 is chosen randomly among the 15 best individuals in the current population. The decision on the number of the last iterations is based on that it is desirable to keep the randomness in the search in the beginning and only fine tune the solution at the very end. The decision on 1000 and 15 individuals are results from preliminary testing performed for the small data instances. The goal of this alteration is to intensify the search for a local optimum in the last stages of the iterative process. The results for GA4iii are shown in table 6.11.

The results for GA4iii are a little bit worse than the results for GA4ii in table 6.10. The total average cost increases 4,0% and only in five data (R2a, R3a, R1b, R2b and R7b) instances the passenger average cost decreases. GA4iii is therefore *rejected*. A possible reason for why the improvement does not improve the results is that the decisions on testing this improvement and the exact numbers, used for the last iterations and individuals to choose parent 1 from, are based on results obtained from the small data instances. The improvement also seems to work well for the smaller data sets. The improvement seems on the other hand not suited for the larger data instances. In the larger instances the 15.000 iterations are not enough for a thorough investigation of the solution space and the fine tuning in the last iterations is waisted. It is better to use them to search for better regions in the solution space. If the number of iterations in GA4iii is increased, I expect that the results for the larger data sets will also improve.

Table 6.11: GA4iii - Parent 1 chosen from the 15 best for the last 1000 iterations.

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
R1a	4774	4420	199	302	1034	524	28	0,95	1	5,46
R2a	17751	13017	370	524	1935	1487	105	1,22	61	11,40
R3a	63615	46519	884	1047	2821	2928	230	1,39	255	21,45
R5a	216819	181416	1807	1352	4358	5668	443	1,68	617	59,00
R9a	337471	316506	3125	1341	3620	6081	184	2,01	1703	40,19
R10a	711705	645698	4942	1820	5058	8535	376	2,01	2758	67,16
R1b	4522	4059	188	296	895	536	4	0,98	1	5,43
R2b	11356	9136	237	549	1690	1166	28	1,06	6	11,56
R5b	115337	80635	961	1363	4287	4377	192	1,41	213	57,35
R6b	204735	156074	1422	1751	5142	6386	223	1,61	431	79,83
R7b	8867	7728	246	483	1292	1050	36	1,17	3	8,01
R9b	169953	142208	1574	1343	3650	5467	177	6,60	529	40,08
R10b	554837	484938	3853	1781	4880	7843	221	1,95	2014	67,30
Total	2421742	2092355	19808	13953	40663	52048	2246	24,04	8590	474,19

6.4.4 Heuristic run twice and total number of iterations constant - GA5

The number of iterations is a constant in the testing performed in this chapter, equal to 15.000 (except for GA2). When having a constant number of iterations one question that arises is how to best utilize these iterations. Until now the iterations have been used solving the problem using one initial population. Another possibility is to run the heuristic twice from the start. In the first run a part of the total iterations are performed and the final population is saved. The final population from that run is mutated drastically, in order to get to another region of the solution space, and the mutated population is used as an initial population for the second run. In the second run a part of the iterations is performed and then the final population along with the final population from the first run are combined. In the combination consists of the better half (the $M/2$ individuals having the lowest fitness values) from each final population from the two runs. The procedure is finished running the remainder of the iterations using the combined population as an initial population. Running the best heuristic encountered so far (GA4ii) in this manner is tested in this section and the improvement is referred to as GA5.

A decision on how many iterations to perform in each run and the last run using the combined population as an initial solution is needed. In order to make that decision graphs of total cost vs. number of iterations are drawn for three randomly chosen runs from GA4ii for data sets R1a, R6b and R9a. Those data sets represent all the types of data sets that are presented in this chapter. The graphs are shown in figure 6.8. When looking at the graphs it can be seen that for all the data sets the cost is high in the beginning and then decreases rapidly and seems to settle, at least for the smallest data set R1a. The heuristic seems to converge to a local optimum. It can also be seen that the rate of decrease in the graphs decreases as the number of iterations increases. After about 5000-6000 iterations the rate of decrease has slowed down and therefore I decided to

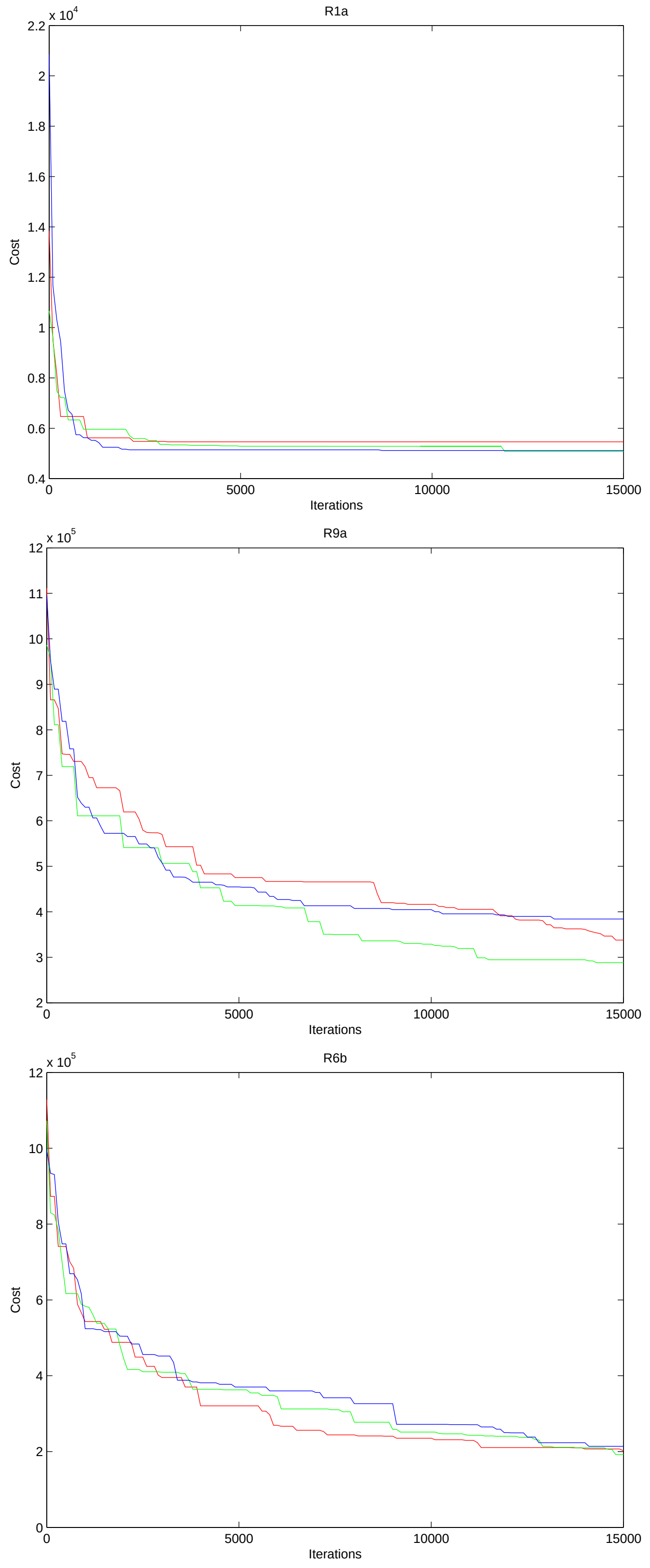

Figure 6.8: Graphs of cost vs number of iterations. The top figure is for R1a, the middle one for R9a and the last one for R6b. In each figure three randomly chosen runs are graphed.

perform 6000 iterations. Then mutate the current population drastically in order to get to another part of the solution space and perform another 6000 iterations using the mutated population as an initial population. After the 12.000 iterations the best individuals from both the populations are combined into a new initial population and the remaining 3000 iterations run normally. The results from GA5 are shown in table 6.12.

Table 6.12: GA5 - Heuristic run twice, constant total number of iterations.

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
R1a	4750	4420	198	306	997	511	42	0,95	4	5,19
R2a	20000	17438	417	542	1933	1728	105	1,46	91	11,41
R3a	73960	61586	1027	1041	2802	3302	170	2	236	21,71
R5a	253451	194694	2112	1364	4382	5669	451	1,68	823	68,60
R9a	368381	345196	3411	1320	3668	5977	176	1,98	1960	49,60
R10a	803899	770412	5583	1792	4943	8248	323	2,02	3433	66,54
R1b	4503	4286	188	298	923	534	9	0,94	0	5,18
R2b	10676	8264	222	551	1654	1146	27	1,06	11	11,37
R5b	125891	93490	1049	1376	4253	5150	374	1,56	230	57,79
R6b	208625	185167	1449	1810	5309	6364	320	1,56	327	77,94
R7b	9006	7065	250	477	1264	1071	22	1,15	3	7,74
R9b	177422	133224	1643	1347	3695	5266	237	1,73	570	46,98
R10b	637350	584775	4426	1783	4872	8241	294	2,01	2261	65,82
Total	2697915	2410019	21974	14006	40695	53207	2551	19,64	9951	495,88

The results are compared to the results from the best improved heuristic encountered so far (GA4ii) presented in table 6.10. The total average cost has increased 15,8% and distance, route duration, ride time, waiting time and time window violation have also increased. When looking at the results for the individual data sets GA5 gives worse results for average passenger cost in all sets except for R1b, R2b and R7b, the small data sets with wide time windows. All in all GA5 gives worse results than GA4ii and is therefore *rejected*. One reason for the bad performance of GA5 is that a total of 9000 iterations for one whole run is not adequate for searching the solution space, even though the best solutions found in another run (of 6000 iterations) are added into the population after 6000 iterations. This holds specially true for the large data instances and the data instances with the narrow time windows (all data instances with an “a” in their names). For the narrow time window data sets it is harder to find solutions that have a low value for the time window violation term in the fitness function. These observations can be supported by figure 6.8, where it can be seen that R1a (24 customers) converges early in the iterations and that R9a (108 customers and narrow TW) has the greatest deviation of the three in final cost results even though it is smaller than R6a (148 customers and wide TW).

Table 6.13: The average results obtained by the best improved heuristic, GA4ii, and results from the best solution for each data set.

	Route duration		Vehicle waiting time				Ride time				CPU
	Average	Best	Average		Best		Average		Best		time
			total	avg	total	avg	total	avg	total	avg	[min]
R1a	1041	1039	252	5,25	260	5,42	477	19,86	310	12,90	5,57
R2a	1969	1994	470	4,90	514	5,36	1367	28,47	1330	27,72	11,43
R3a	2779	2781	292	2,03	301	2,09	3081	42,79	2894	40,20	21,58
R5a	4250	4274	500	2,08	527	2,20	5099	42,49	4837	40,30	58,23
R9a	3597	3526	94	0,44	32	0,15	6251	57,88	6719	62,21	40,78
R10a	5006	5025	315	1,09	246	0,86	8413	58,42	8341	57,92	65,98
R1b	907	928	143	2,98	164	3,42	630	26,24	549	22,89	5,46
R2b	1719	1710	198	2,06	162	1,69	1214	25,30	1300	27,07	11,72
R5b	4296	4336	552	2,30	568	2,37	4615	38,46	4720	39,33	58,93
R6b	5309	5227	630	2,19	513	1,78	6134	42,59	6397	44,42	81,23
R7b	1299	1316	102	1,41	128	1,78	990	27,50	784	21,76	8,29
R9b	3679	3676	147	0,68	177	0,82	5362	49,65	5358	49,61	44,66
R10b	4733	4678	113	0,39	85	0,29	7969	55,34	8119	56,38	66,41
Total	40584	40508	3808	27,81	3678	28,21	51600	514,99	51657	502,72	488,61

6.5 The best improvement

The initial heuristic and six improvements have been tested. The improved heuristic, named GA4ii, gives the best results in the testing. GA4ii decreases average cost of 23,8% from the initial heuristic GA1, using customers choice of weights, while the CPU time has gone up 6,9%. The reason for the increased CPU time is that in GA4ii duplicate offsprings are mutated (if a duplicate is produced in a crossover it is mutated and the routing algorithm is run twice for the offspring) and parents are switched if parent 2 represents the better parent. Both these changes to GA1 add a constant time to the CPU time but the number of times is random. The new method of creating the template does not add to the computational time.

6.6 Comparison with Cordeau and Laporte results

In this section the results obtained using the best improved heuristic is compared to the results obtained by Cordeau and Laporte. The average results from the best improved heuristic (GA4ii) is presented in table 6.13 along with the results for the best solution obtained for each data set (i.e. the solution with the lowest cost). Those are some of the same results as in table 6.10 but additionally vehicle waiting time is reported. The vehicle waiting time is not reported in earlier tables since it is not a part of the objective to minimize the vehicle waiting time but the passengers waiting time, which is reported in the tables earlier in the chapter. Cordeau and Laporte on the other hand report this waiting time and therefore it is included in table 6.13. The best solutions results are not that different from the average results, route duration and waiting time little lower and ride time little higher.

Table 6.14: Results obtained by Cordeau and Laporte.

	Route	Vehicle wait. time		Ride time		CPU
	duration	total	average	total	average	[min]
R1a	881	211	4,4	1095	45,62	1,90
R2a	1985	724	7,54	1977	41,18	8,06
R3a	2579	607	4,22	3587	49,82	17,18
R5a	3870	833	3,47	6154	51,3	46,24
R9a	3155	323	1,5	5622	52,05	50,51
R10a	4480	721	2,5	7164	49,75	87,53
R1b	965	321	6,68	1042	43,4	1,93
R2b	1565	309	3,22	2393	49,86	8,29
R5b	3596	606	2,52	6105	50,87	54,33
R6b	4072	449	1,56	7347	51,02	73,70
R7b	1097	129	1,79	1762	48,94	4,23
R9b	3249	487	2,26	5581	51,68	51,28
R10b	4041	362	1,26	7072	49,11	92,41
Total	35537	6082	42,92	56900	634,6	497,59

Cordeau and Laporte performed 10^4 to 10^5 iterations in order to get their best results which are presented in table 6.14. In the testing in this project 15.000 iterations are performed. The total CPU time is however comparable. Cordeau and Laporte use a Pentium 4, 2 GHz computer in their CPU measurements. Recall that the experiments in this project are performed on an Intel Celeron CPU 2 GHz computer. It can be pointed out again that on the average 99% of the CPU time in this project is by the routing heuristic.

The objective function used by Cordeau and Laporte is:

$$v(S) = v_1(S) + \alpha v_2(S) + \beta v_3(S) + \gamma v_4(S) + \tau v_5(S) \tag{6.2}$$

where $v(S)$ is the objective value for solution S, $v_1(S)$ denotes the total routing cost of the vehicles, $v_2(S)$ denotes total load violation, $v_3(S)$ denotes total route duration violation, $v_4(S)$ denotes total time window violation and $v_5(S)$ denotes total ride time violation. α, β, γ and τ are a self-adjusting weights. The values of these weights changes in the iterations and no approximate values are given in the paper. The method used to calculate the total routing cost is not specified either in the paper. It is therefore impossible to compare the cost obtained by the two solution methods.

The other results can be compared to some extent. They are; route duration, ride time and vehicle waiting time. It can be seen that the route duration is higher in GA4ii than in Cordeau and Laporte. The other two results, which are related to customer service, ride time and vehicle waiting time are on the other hand lower in GA4ii than in the results obtained by Cordeau and Laporte. One reason for these results is that the weights are set to represent the customers choice in this project so there is an emphasis on customer service factors.

6.7 Summary

Table 6.15: Summary of results obtained by the initial heuristic and the improvements. The upper part presents the absolute results for each heuristic and the lower part presents the results relative to GA4ii. A value lower than 1 indicates a lower result than in GA4ii and a higher value than 1 indicates a higher result.

	Cost		Pass.	Dist-	Route	Ride	Wait.	Avg	TW	CPU
	avg	best	cost	ance	durat.	time	time	load	viol	[min]
GA1	3058453	2723812	25186	14048	40971	55855	3130	20,06	11687	449,44
GA2	8975233	6766929	75835	14175	43518	1762329	7828	22,79	39959	481,36
GA3	2899023	2577013	23619	13964	40441	54285	2715	19,86	10946	465,49
GA4i	2490574	2051715	20361	13930	40574	52644	2493	19,71	8703	485,44
GA4ii	2330280	2021580	19206	13976	40584	51600	2109	19,18	8088	480,28
GA4iii	2421742	2092355	19808	13953	40663	52048	2246	24,04	8590	474,19
GA5	2697915	2410019	21974	14006	40695	53207	2551	19,64	9951	495,88
GA1	1,31	1,35	1,31	1,01	1,01	1,08	1,48	1,05	1,44	0,94
GA2	3,85	3,35	3,95	1,01	1,07	34,15	3,71	1,19	4,94	1,00
GA3	1,24	1,27	1,23	1,00	1,00	1,05	1,29	1,04	1,35	0,97
GA4i	1,07	1,01	1,06	1,00	1,00	1,02	1,18	1,03	1,08	1,01
GA4ii	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
GA4iii	1,04	1,04	1,03	1,00	1,00	1,01	1,06	1,25	1,06	0,99
GA5	1,16	1,19	1,14	1,00	1,00	1,03	1,21	1,02	1,23	1,03

The initial heuristic along with six improvements have been tested and the results reported in this chapter and summarized in table 6.15. In the upper part of the table presents the total columns for each heuristic and the lower part presents the proportion of each heuristics results in comparison to GA4ii. A value lower than 1 indicates a lower result than obtained by GA4ii, i.e. a better result for all columns except average load. A value higher than 1 indicates a higher result than for GA4ii, i.e. a worse result for all columns except average load. The distance, passenger cost, route duration, and ride time for six of the seven results introduced earlier are also summarized in figure 6.9. In the figure the results for GA2 are omitted, since GA2 gives much worse results for cost than the other ones.

In the table and the figure it can be seen that GA4ii gives the best results in all cases except for average load. The cost, ride time, waiting time and time window violation are all influenced by the improvements while the different heuristics give a near constant results for distance and route duration. The largest improvements for GA4ii are decreasing average cost of 23,8% and time window violation of 30,8% compared to GA1. In the other categories the results are similar but the CPU time has gone up 6,7%. The GA4ii proves to be superior to GA1 and is selected as the best solution method found in this project.

Some of the results that are presented in this chapter came as a big surprise to me. For example, I had pictured that the result would improve when the size of the population in the genetic algorithm is increased. This is not the case. The main reason for that is that for larger populations, a larger number of iterations are needed in order to perform

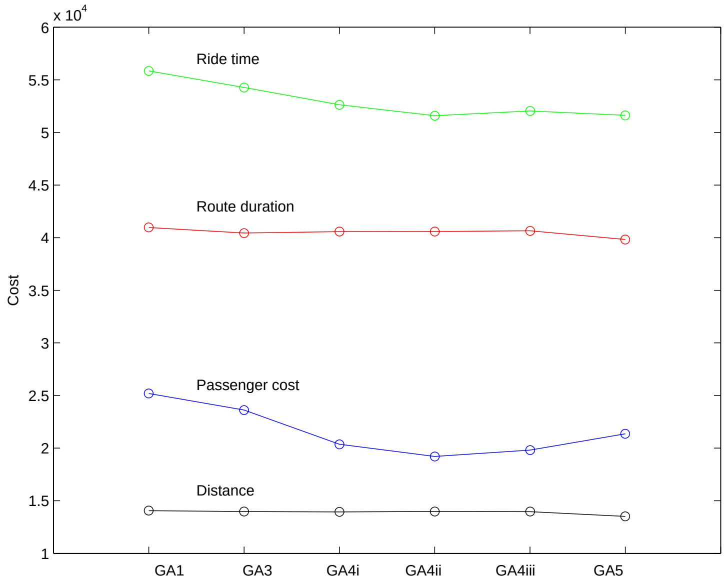

Figure 6.9: Comparing the cost, distance, route duration and ride time for GA1 and GA3 to GA5.

a local search for an optimum. Another thing that surprised me, is the behaviour of the parameters that are tuned in the GA. No explicit patterns are present and it makes it hard to select a good combination of the parameter values.

Three of the proposed improvements tested in the chapter do not improve the results, which is not what I expected. Firstly, increasing the generation size from 1 to M gives much worse results using similar CPU times as the initial heuristic. In order to obtain similar CPU times it is necessary to decrease the number of iterations from 15.000 to 300, which turns out to be non sufficient for searching the solution space and converging to a local optimum. Secondly, selecting parent 1 among the 15 best individuals for the last 1000 iterations in order to get a better local search in the final stages of the search for a local optimum, does not improve the results. One reason for its failure to improve results is that decreasing the number of iterations with random search, resulted in a worsened investigation of the solution space, especially for the larger data sets. Another reason is that this improvement is the last of three improvements tested to reduce randomness in the heuristic. The first two are accepted and now the randomness is less than before. Perhaps it is now not desirable to reduce the randomness further and if the three random reducing improvements are tested in a different order this improvement would be accepted. Thirdly, running the heuristic twice does not give better results, especially for the small data sets. The reason is again that decreasing the number of iterations for a single run as is performed here, will worsen the results.

A best heuristic is selected and compared to the results obtained by Cordeau and Laporte. It is not possible to compare the costs but instead route duration, vehicle waiting time and ride time are compared. The results show that the best heuristic gives comparable results to the results obtained by Cordeau and Laporte.

The JAVA source code for the heuristic developed in this project is given in Appendix A.

Chapter 7

Conclusion

In this project the focus has been on solving the static dial-a-ride problem using a solution procedure not previously used for solving the DARP. Even though the project is mainly theoretical realistic considerations from a Danish transportation operator have been included.

The project starts by describing the DARP both in words, by examples and by presenting the mathematical formulation of DARP. The mathematical formulation is based on existing literature but a few additions are presented in this project, cost of waiting time with customers and constraints on maximum route duration and ride time.

An overview of the relevant literature that has given an inspiration to this project is presented. This literature is a part of the background for choosing the method to solve the formulated dial-a-ride problem.

The solution method chosen for solving the DARP is the classical two-phase approach cluster-first, route-second. First, customers are clustered and the number of clusters equals the number of available transportation vehicles. Second, a route is developed for all the vehicles. The route is an ordered list of the pickup and drop off locations of the customers along with arrival and departure times at each location. The clustering is solved using the genetic algorithm and the routing is solved using the modified space-time nearest neighbour heuristic developed by Baugh et al. [2]. In the project, the modified space-time nearest neighbour heuristic is extended to include the constraints present in the formulated DARP. The extensions are to include the hard depot constraints, the soft maximum route duration and maximum ride time constraints. Also included are service times, waiting times with passengers and excess ride times. An initial heuristic is constructed and six improvements suggested.

The initial heuristic along with the suggested improvements are implemented in JAVA. The initial heuristic is tuned and tested using 16 randomly generated data sets made by Cordeau and Laporte [4]. 3 of the 16 data sets are chosen for tuning and the remaining 13 for examining the behaviour of the heuristic. Three values for four parameters in the genetic algorithm are selected based on the cost and CPU time results for solutions obtained for every combination of the factors. In the initial heuristic it is possible to have different emphasis on the factors present in the objective function by adjusting the values of weights. The results of selecting the weights as a customer and transportation operator are presented.

A number of potential improvements are tested and define the final "product" of this project. The results from the final "product" heuristic found in this project are compared to the results given by Cordeau and Laporte. The comparison focuses on route duration, ride and vehicle waiting times. The comparison showed that Cordeau and Laporte obtain better results for route duration whereas th final"product" heuristic obtains better results in the other two categories. The work presented by Cordeau and Laporte can be considered state-of-the-art within the OR, and my best results are comparable to theirs.

New ideas have been tried in the solution method, which were not encountered in the literature. One is the method for selecting the first customer in a route, which is based on selecting the customer with the earliest latest pickup time. The second is the modification of the roulette wheel method in the genetic algorithm to fit a minimization problem, where ideas are taken from physics.

Several ideas about improvements were left untested due to lack of time. For example to use ranking when selecting parents in the genetic algorithm and to experiment with different crossover and mutation operators. I would also have liked to implement a new routing algorithm. I was not pleased with the modified space-time nearest neighbour heuristic. It was described as a fast heuristic but when comparing the CPU times with those obtained by Cordeau and Laporte that cannot be the case. They are able to perform up to 100.000 iterations with the same amount of CPU time as my "best" heuristic in this project uses when performing 15.000 iterations on almost the same computer. About 99% of the CPU time of the "best" heuristic is used by the routing algorithm and it would be possible to improve the performance if a new routing algorithm was implemented. A possible alternative from using the modified space-time nearest neighbour heuristic is to use enumeration, i.e. to check all possible combinations of the stops and select the combination with the least cost. Another possible alternative is to implement the Lin-Kernighan algorithm [2].

The overall conclusion of this project is that a new solution method that is able to solve the DARP has been implemented and tested. It is possible to adjust the weights of seven factors in the cost calculations. That makes it possible to take different points of a view of how a dial-a-ride system should look like. The method can therefore be used to examine the influence of different strategies and different emphasis in the DAR systems. The new solution method has achieved solutions comparable to the current state-of-the-art methods.

Bibliography

[1] R. Borndorfer, M. Grotschel, F. Klostermeiner and C. Kuttner. *Telebus Berlin: Vehicle Scheduling in a Dial-a-Ride System.* Konrad-Zuse-Zentrum fur Informationstechnik Berlin, 1997.

[2] J.W. Baugh jr., D.K.R. Kakivaya and J.R. Stone. *Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing.* Engineering Optimization, 30(2):91 - 124, 1998.

[3] J.F. Cordeau, M. Gendrau and G. Laporte. *A tabu search heuristic for periodic and multi-depot vehicle routing problems.* Networks 30, 105 - 119, 1997

[4] J.F. Cordeau and G. Laporte. *A tabu search heuristic for the static multi-vehicle dial-a-ride problem.* Transportation Research, Part B, 37:579 - 594, 2003.

[5] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to Algorithms.* McGraw-Hill Book Company, 2000.

[6] P. Healy and R. Moll. *A new extension of local search applied to the dial-a-ride problem.* European Journal of Operational Research, 83(1):83 - 104, 1995.

[7] F. S. Hiller and G. J. Lieberman. *Introduction to Operations Research*, sixth edition. McDraw-Hill Publishing Company, 1995.

[8] I. Ioachim, J. Desrosiers, Y. Dumas, M. M. Solomon and D. Villeneuve. *A request clustering algorithm for door-to-door handicapped transportation.* Transportation Scinece, 29(1):63 - 78, 1995.

[9] J.J. Jaw, A.R. Odoni, H.N. Psaraftis and N.H.M. Wilson. *A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows.* Transportation Research, Part B (Methodological), 20B(3):243 - 257, 1986.

[10] W.R. Jih, C.Y. Kao and F.Y.J. Hsu. *Using Family Competition Genetic Algorithm in Pickup and Delivery Problem with Time Window Constraints.* In Proceedings of the 2002 IEEE International Symposium on Intelligent Control, Vancouver, Canada, 496 - 501, 2002.

[11] R.M. Jørgensen. *Dial-a-Ride.* Doctor's thesis, Technical University of Denmark, 2002.

[12] O.B.G. Madsen, H.F. Ravn and J.M. Rygaard. *A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities and multiple objectives.* Annals of Operation Research, 60:193 - 208, 1995.

[13] F.B. Pereira, J. Tavares, P. Machado and E. Costa. *GVR: a New Genetic Representation for the Vehicle Routing Problem.* Artificial Intelligence and Cognitive Science. 13th Irish Conference, AICS 2002. Proceedings (Lecture Notes in Artificial Intelligence Vol. 2464), 95 - 102, 2002.

[14] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* McGraw-Hill International Limited UK, 151 - 188, 1995.

[15] S.M. Sait and H. Youssef.*Iterative Computer Algorithms with Application in Engineering: Solving Combinatiorial Optimization Problems.* IEEE Compture Society, 109 - 167, 1999.

[16] P. Toth and D. Vigo. *Heuristic algorithms for the handicapped persons transportation problem.* Transportation Science, 31(1):60 - 71, 1997.

[17] P. Toth and D. Vigo. *Heuristic algorithm for the symmetric and asymmetric vehicle routing problems with backhauls.* European Journal of Operational Research Issue, 113(3), 528 - 543, 1999.

Appendix A

JAVA source code

The JAVA source code for the heuristic developed in this project is given as follows:

```
import java.util.*;
import java.io.*;

class GA5{
    public static void main (String[] args) {

    long cputime = System.currentTimeMillis();
    long routetime=0, rtime;

    //Define parameters of the algorithm
    String file = "pr16";      //Name of data file
    final int M = 50;         //Population size

    final int G = 15000;         //Number of generations, iterations
    int Z = (int)M/10;          //Worst individuals in population
    if (Z==0) Z=1;              //At least 1
    System.out.println("Z="+Z+" M="+M+" G="+G+" "+file);
    final float pm = 1;         //Mutation rate * 100
    final float big = 3.3E+38f; //big number

    float just=60;             //likely hood of choosing genes from p1
    //float fjust=90;          //max likely hood after iterations

    //read data file + definations
    Request req = new Request(file);
    Car vehicle = new Car(file);
    int car = vehicle.getNOCar();         //number of cars available
    Route26042 ro = new Route26042(file);
    final int stops = req.getStops();     //number of stops
    //I assume there are always at least 2 stops
    //and there is an even number of them
```

```
Chromo crowd = new Chromo(M, car, stops);

//make initial population
crowd.createRandInitPop();
byte[][][] population = crowd.getPopulation();
byte[][][] population1 = new byte[M][car][stops/2+1];

//Evaluate initial population

byte[]clust = new byte[stops/2+1];  //include all cust + depot
float clustercost;
float[] crowdcost = new float[M];
float[] crowdcost1 = new float[M];

//cost for each individual in population
//System.out.println("clustercost ");
for(int i=0;i<M;i++){
    clustercost = 0;
    for(int j=0;j<car;j++){
    clust = crowd.getCluster(i,j);
    rtime = System.currentTimeMillis();
    ro.getCustomers(clust,stops/2);
    clustercost=clustercost+ro.getFinalcost();
    rtime = System.currentTimeMillis()-rtime;
    routetime = routetime + rtime;
    }//for ends
    crowdcost[i]=clustercost;
    //System.out.print(crowdcost[i]+" ");
}//for ends


//Genetic algorithm, G iterations
byte[][]offspring = new byte[car][stops/2+1];
float offcost;
float mutecost = big;

for(int i=0;i<G;i++){

    if(i==6000){
    for(int tel1=0;tel1<M;tel1++){
        crowdcost1[tel1] = crowdcost[tel1];
        for(int s=1;s<stops/2+1;s++){
        int sum=0,t=-1;
        int[] ones = new int[car];
        for(int c=0;c<car;c++){
            population1[tel1][c][s] = population[tel1][c][s];
            population1[tel1][c][0] = 1;
        if(population[tel1][c][s]==0){
        population[tel1][c][s]=1;
        t++;
        ones[t] = c;
        }
        else population[tel1][c][s]=0;
        sum = sum+population[tel1][c][s];
    }//for

    if(sum<1) population[tel1][randInt(car)][s]=1;

    if(sum>1){
        for(int p=1;p<sum;p++){
        int rand=randInt(sum);
        while (ones[rand]==-1)rand=randInt(sum);
        population[tel1][ones[rand]][s]=0;
        ones[rand]=-1;
        }//for
    }//if sum

    }//for s

    //Evaluate the new mutated population
    clustercost = 0;
    byte[] line = new byte[stops/2+1];
    for(int kk=0;kk<car;kk++){
    for(int mm=0;mm<stops/2+1;mm++)
        line[mm] = population[tel1][kk][mm];
    rtime = System.currentTimeMillis();
    ro.getCustomers(line,stops/2);
    clustercost=clustercost+ro.getFinalcost();
    rtime = System.currentTimeMillis()-rtime;
    routetime = rtime+routetime;
    }//for ends

    crowdcost[tel1]=clustercost;

}//for tel1
printSolution(M,i,crowdcost);
}//end (if i== 6000)

if(i==12000){
Help h = new Help();

float[] cc1=new float[crowdcost1.length];
float[] cc2=new float[crowdcost.length];
```

```
for(int ix=0;ix<crowdcost1.length;ix++){
    cc1[ix]=crowdcost1[ix];
    cc2[ix]=crowdcost[ix];
}
int[]cost1 = h.bubble(cc1);
int[]cost2 = h.bubble(cc2);

for(int t2=M/2;t2<cost2.length;t2++){
    int t1 = t2-M/2;
    crowdcost[cost2[t2]]=crowdcost1[cost1[t1]];
    population[cost2[t2]] = population1[cost1[t1]];
}

}//end if(i==12000)

//select individual stochastically from population
//as parent 1
int rand;
int parent1;

rand = crowd.rand_int(100000000);
parent1 = parenta(M,crowdcost,rand);

//select random individual from population as parent 2
//different than parent 1

int parent2;

do {
parent2 = crowd.rand_int(M);
}while(parent2==parent1);

//if parent2 is better than parent1 then switch
if(crowdcost[parent1]>crowdcost[parent2]){
int parentx;
parentx=parent1;
parent1=parent2;
parent2=parentx;
}

byte[][]p1 = crowd.getIndividual(parent1);
byte[][]p2 = crowd.getIndividual(parent2);

//offspring generated, crossover parent 1 and 2
offspring =
crowd.cross_over2(p1,p2,just);
```

```
//evaluate offspring
byte[] line = new byte[stops/2+1];
offcost = 0;
for(int kk=0;kk<car;kk++){
for(int mm=0;mm<stops/2+1;mm++)
    line[mm] = offspring[kk][mm];
rtime = System.currentTimeMillis();
ro.getCustomers(line,stops/2);
offcost=offcost+ro.getFinalcost();
rtime = System.currentTimeMillis()-rtime;
routetime = rtime+routetime;
}//for ends

//if offsprings cost same as parent 1 then mutate
int key=0;
if(Math.abs(offcost -crowdcost[parent1])< 1E-5f)
key=80;

//Mutation takes place with probability pm
rand = crowd.rand_int(100);
if (rand>=0 && rand<=pm || key==80){
key = 0;
crowd.mutation(offspring);

//evaluate mutated offspring
mutecost = 0;
for(int kk=0;kk<car;kk++){
    for(int mm=0;mm<stops/2+1;mm++)
    line[mm] = offspring[kk][mm];
    rtime = System.currentTimeMillis();
    ro.getCustomers(line,stops/2);
    mutecost=mutecost+ro.getFinalcost();
    rtime = System.currentTimeMillis()-rtime;
    routetime = rtime+routetime;
}//for ends
}//if ends

if(mutecost < big) offcost = mutecost;
mutecost = big;
//}//for j ends

//insert offspring for random individual among
//the worst in population
//Z is proportional to M
float[] worst = new float[Z];  //cost of worst individuals
int[] wno =  new int[Z];
//indices of corresponding individuals
```

```
    for(int z=0;z<worst.length;z++){
    worst[z] = crowdcost[z];
    wno[z] = z;
    }//for ends

    //small is the index of smallest element in worst
    int small = findSmall(worst,M);

    for(int t=Z;t<crowdcost.length;t++){
    if(worst[small] < crowdcost[t]) {
        worst[small] = crowdcost[t];
        wno[small] = t;
    }
    small = findSmall(worst,M);
    }//for ends

    //choose random individual among the worst with the higest cost
    int random = crowd.rand_int(wno.length);
    int b = wno[random];

    //offspring replaces the chosen individual
    //in the current population
    crowdcost[b] = offcost;
    for(int c=0;c<car;c++){
    for(int s=0;s<stops/2+1;s++){
        population[b][c][s]=offspring[c][s];
    }
    }

}//for i ends

cputime = System.currentTimeMillis()-cputime;

//print solution
int small = findSmall(crowdcost,M);
byte[] line = new byte[stops/2+1];
float[]ddist = new float[car];
float[] samtalsridet = new float[car];
float[]rodur = new float[car];
float[]tw = new float[car];
float[]waittime = new float[car];
float[]bidtimifylki = new float[car];
int customer = stops/2;
System.out.print(file+" "+customer);
for(int kk=0;kk<car;kk++){
    for(int mm=0;mm<stops/2+1;mm++)
```

```
        line[mm] = population[small][kk][mm];
        ro.getCustomers(line,stops/2);
        int sum=0;
        int[]order = ro.getRoute();
        for(int k3=0;k3<order.length;k3++)sum = sum+order[k3];
        int ld=0;
        int[] load = ro.getLoad();
        for(int k6=0;k6<load.length;k6++)ld = ld + load[k6];
        int leng = load.length;
        float prufa = leng-2;
        System.out.print(" sum "+ld);
        System.out.print(" "+prufa);
        ddist[kk]=ro.getDist();
        float[] ride = ro.getRidetime();
        for(int je=0;je<ride.length;je++){
        if(je%2==1)
            samtalsridet[kk]=samtalsridet[kk] + ride[je];
        }
        float[]time= ro.getTime();
        rodur[kk]=ro.getRouteDuration();
        float[]window = ro.getTimeWindowsViol();
        for(int k7=0; k7<window.length;k7++)tw[kk]=tw[kk]+window[k7];
        float summa = 0f;
        float[]bidtimi = new float[load.length];
        float[]wait = ro.getWaitingTime();
        for(int k8=1; k8<wait.length;k8++){
            waittime[kk] = waittime[kk] + wait[k8];
            bidtimi[k8]=load[k8-1]*wait[k8];
            summa = summa + bidtimi[k8];
        }
        bidtimifylki[kk]=summa;
    }//for ends

    float samtals=0;
    System.out.println("Distance for");
    for(int tel=0; tel<ddist.length;tel++){
        //System.out.print(" route "+tel+" is "+ddist[tel]);
        samtals =samtals+ddist[tel];
    }
    System.out.println();
    System.out.print(" dist "+samtals);
    System.out.println();
    samtals=0;
    System.out.println("Ridetime for passangers in");
    for(int tel=0; tel<samtalsridet.length;tel++){
        System.out.print(" route "+tel+" is "+samtalsridet[tel]);
        samtals =samtals+samtalsridet[tel];
    }
    System.out.println();
    System.out.print(" ride "+samtals);
    System.out.println();
    samtals=0;
    System.out.println("Route duration in");
    for(int tel=0; tel<rodur.length;tel++){
        System.out.print(" route "+tel+" is "+rodur[tel]);
        samtals =samtals+rodur[tel];
    }
    System.out.print(" route "+samtals);
    samtals=0;
    System.out.println("Timewindows violation in");
    for(int tel=0; tel<tw.length;tel++){
        System.out.print(" route "+tel+" is "+tw[tel]);
        samtals =samtals+tw[tel];
    }
    System.out.println();
    System.out.print(" tw "+samtals);
    samtals=0;
    System.out.println("Waiting times in");
    for(int tel=0; tel<waittime.length;tel++){
        System.out.print(" route "+tel+" is "+waittime[tel]);
        samtals =samtals+waittime[tel];
    }
    System.out.println();
    System.out.print(" wait "+samtals);
    System.out.println();
    samtals=0;
    System.out.println("Waiting times in");
    for(int tel=0; tel<bidtimifylki.length;tel++){
        System.out.print(" route "+tel+" is "+waittime[tel]);
        samtals =samtals+bidtimifylki[tel];
    }
    System.out.println();
    System.out.print(" bid "+samtals);
    System.out.println();
    System.out.print(" M="+M+" G="+G+" pm="+pm+" Z="+Z+" "+file);
    System.out.print(" cost "+crowdcost[small]);
    System.out.print(" CPU "+cputime);
    System.out.println("min: "+cputime/60000);
    System.out.print(" Routingrunningtime "+routetime);
    System.out.println
    ("Difference: "+(cputime-routetime)+" or ca "+
    ((cputime-routetime)*100/cputime)+"%");
    System.out.println("Number of customers is "+stops/2);
    System.out.println();
```

```
   }//main ends

   //print solution
   public static void printSolution(int M,int i,float[]crowdcost){
   int small = findSmall(crowdcost,M);
   System.out.print(i+" , "+crowdcost[small]);
   float[]cr = new float[crowdcost.length];

   for(int ij=0;ij<cr.length; ij++)
       cr[ij]=-crowdcost[ij];

   small = findSmall(cr,M);
   System.out.println(" , "+crowdcost[small]);
   System.out.println();
   }

   //finds smallest element in an array vec and returns
   public static int findSmall(int[] vec, int M){
   int small = 2*M, t;
   int sm = 2000000000, sm1;

   for(t=0;t<vec.length;t++){
       sm1=sm;
       sm = Math.min(sm,vec[t]);
       if(sm != sm1) small = t;
   }//for ends

   return(small);
   }//findSmall ends

   public static int findSmall(float[] vec, int M){
   int small = 2*M, t;
   float sm = 3.0E+38f, sm1;

   for(t=0;t<vec.length;t++){
       sm1=sm;
       sm = Math.min(sm,vec[t]);
       if(sm != sm1) small = t;
   }//for ends

   return(small);
   }//findSmall ends

public static int parenta(int M, float[]cost, int rand){
   //select individual stochastically from population
   //as parent 1
   float S=0;
```

```
    int B=100000000;
    for (int index=0;index<M;index++)
        S = S + 1/cost[index];

    int[] probability = new int[M];
    probability[0] = (int)(B/(cost[0]*S));
    //probability*B of selecting ind 0 as parent 1
    int parent1 = M-1;  //default if no patent is chosen,
    //because of numeration errors

    for(int index=1;index<M;index++){
        probability[index] =
        probability[index-1] + (int)(B/(cost[index]*S));
    }

    if(rand < probability[0]) parent1 = 0;
    for (int index=1;index<M;index++){
        if(rand>=probability[index-1] && rand<probability[index])
        parent1 = index;
    }//for ends
    return(parent1);
    } //parenta ends

    public static int parentb(int M, float[]cost, int rand){
    int small=0;
    float[] cro = new float[cost.length];
    for(int ij=0;ij<cost.length;ij++)
        cro[ij] = cost[ij];

    for(int ij=0;ij<=rand;ij++){
        small=findSmall(cro,M);
        cro[small]=3.3E36f;
    }

    return(small);
    }//parentb ends

    public static int randInt(int L){ return((int)(Math.random()*L));}

}//class ends

class Help{
    int[] bubble(float[] matrix){
    int[] place = new int[matrix.length];
    for(int t=0;t<place.length;t++)
        place[t] = t;
    for(int t1=matrix.length-1;t1>0;t1--){
        for(int t2=0;t2<t1;t2++){
        if(matrix[t2]>matrix[t2+1]){
            float temp = matrix[t2];
            int tplace = place[t2];
            matrix[t2]= matrix[t2+1];
            matrix[t2+1]=temp;
            place[t2]=place[t2+1];
            place[t2+1] = tplace;
        }
        }
    }
    return place;
    }//bubble ends
}//class ends


class Route26042 {

    private int[] v;         //customers in cluster
    private int[] cs;        //customers served
    private int[] cv;        //customers in vehicle
    private int[] cn;        //customers not jet served

    int MMM;                //total number of customers
    String filename;
    int n;                 //total number of customers in all clusters
    int cun;               //number of customers in cluster
    int speed = 1;         //travelling time equal to E. dist
    float w1=1,w2;     //(2)weight total route dur and time windows viol
    float w3,w4;        //weight on ride time and route duration viol
    float w5=3,w6=1;       //5(0)weight on ex ridetime and waiting time
    float w7=8;            //weight on distance
    int[] ord;             //array that holds the order of cust in route
    int[] load;            //array with number of customers in vehicle after
                           //a node has been serviced

    Request r;
    Distance d;
    Car car;
    float[] ctime, wtime;

    public Route26042(String file){filename = file;}

    public void giveOrder(int[] order, int nn, float begin){
    ctime = new float[order.length];
    ctime[0]=begin;
```

```
wtime = new float[order.length];
load = new int[order.length];
ord = new int[order.length];
ord = order;
cun = order.length/2-1;
n = nn;
}

public int[] getRoute(){return(ord);}

public float getFinalcost(){
if (ord.length > 2){
    float finalcost = ordcost(ord);
    return(finalcost);
}
else return(0);
}


public int[] getLoad(){
return(load);
}


public float[] getTime(){
   if (ord.length > 2){
    float cost = getFinalcost();
}
   return(ctime);
}

//returns array with violated times
public float[] getTimeWindowsViol(){
//float cost = getFinalcost();
float[]out = new float[ord.length];
for(int i=0;i<ord.length;i++){
   if(ctime[i]>=r.getLTimeWindow(ord[i])+r.getServicetime(ord[i])&&
      ctime[i]<=r.getUTimeWindow(ord[i])+r.getServicetime(ord[i]))
    out[i]=0;
    else if
    (ctime[i]<
     r.getLTimeWindow(ord[i])+r.getServicetime(ord[i]))
    out[i]=
     ctime[i]-r.getLTimeWindow(ord[i])-r.getServicetime(ord[i]);
    else out[i]=
     ctime[i]-r.getUTimeWindow(ord[i])-r.getServicetime(ord[i]);
}//for ends
```

```
return(out);
}


//returns duration of a route
public float getRouteDuration(){
return(ctime[ctime.length-1]-ctime[0]);
}

//returns ride times of all customers in one route
public float[] getRidetime(){

float[]ride = new float[2*cun];
float cost=ordcost(ord);
int tel=0;
for(int i1=1;i1<ord.length-1;i1++){
    if(ord[i1]<=n){
    for(int i2=2;i2<ord.length-1;i2++){
        if(ord[i2]==ord[i1]+n){
        ride[tel] = ord[i1];
        ride[tel+1]=ctime[i2]
            -ctime[i1]-r.getServicetime(ord[i1]);
        tel=tel+2;
        }//if
    }//for
    }//if
}//for

return(ride);
}//getRidetime

//returns waiting times of all nodes in one route
public float[] getWaitingTime(){
return wtime;
}

//returns total distance between nodes in ord
public float getDist(){
float dist=0;
for(int i=0;i<ord.length-1;i++)
    dist = dist+d.getDistance(ord[i],ord[i+1]);
return(dist);
}

public void getCustomers(byte[] clu, int MM){
//clu inholds customers to be routed
w2=MM;
```

```
w3=MM;
w4=MM;


cun = 0;
for (int kk=1;kk<clu.length;kk++)
    cun=cun+clu[kk];              //number of customers in cluster
n = clu.length-1;    //total number of customers, depot not included

r = new Request(filename);
d = new Distance(filename);
car = new Car(filename);


//if there are any customers in cluster then continue, otherwise
//return depot to depot route

if(cun > 0){
    v = new int[cun];
    //v has the number of the customers in the cluster
    int count = 0;
    for (int i=1;i<clu.length;i++){
    if (clu[i]==1){
        v[count]=i;
        count++;
    }//if ends
    }//for ends

    route();

}//if ends
else {
    ord = new int[2];
    ord[0] = 0;
    ord[1] = 0;
    ctime = new float[2];
    ctime[0] = 0;
    ctime[1] = 0;
}
}//getCustomers ends


// -------------------- ROUTE ------------------------ //


//generate route for a cluster, algorithm taken from Baugh et al
public void route(){
```

```
ord = new int[2*cun+2];        //order of customers in solution
load = new int[2*cun+2];
ctime = new float[2*cun+2];  //time at which car has
                             //finished servicing the nodes,
wtime=new float[2*cun+2];
int nnode=-1, cnode=0;         //newnode and current node
float ttime=20000, mincost;

//order as in resulting route

int sumcs=0;       //sum: cs - customers that have been serviced
int sumcn=0;       //sum: cn - customers not in vehicle
int sumv=0;        //sum: v  - all customers numbers in cluster

int[] firstcust=new int[v.length];
//first customer in route, cust no
int[] firstno=new int[cun];
//number of customer in route

cs = new int[cun];
cv = new int[cun];
cn = new int[cun];
int[]N4 = new int[4]; //4 closest nodes to be considered as next node

//*****************************************************

//find cust in v with earliest latest pick-up time
//as first customer in route
int first=-5, no=-5;
float mini=3.3E38f;
float[]ultw = new float[v.length];

for(int re=0;re<v.length;re++){
    if(v[re]>n/2) ultw[re]=r.getUTimeWindow(v[re]);
    else ultw[re]=r.getUTimeWindow(v[re]+n)-
         d.getDistance(v[re],v[re]+n)/speed-
         r.getServicetime(v[re]);
    if(mini>ultw[re]){mini=ultw[re]; no=re; first=v[re];}
}//for


//values set as -1 to indicate no customer for cs and cv, i.e.
//no customer is in vehicle or has been serviced in the begining,
//all customers number set into cn, customers not serviced

for(int i=0;i<v.length;i++){cn[i]=v[i]; cs[i]=-1; cv[i]=-1;}
```

```
cv[no] = first;  //first customer added to vehcile
cn[no] = -1;         //first customer deleted from cn
ord[0]= 0;                    //start in depot
ord[1] = first;           //then to first customer
load[0] = r.getLoadChange(0);
load[1] = load[0]+r.getLoadChange(first);

//sums calulated, used as stopping criterias in while loop
for(int i=0;i<cs.length;i++){
    sumcs = sumcs + cs[i];
    sumcn = sumcn + cn[i];
    sumv = sumv + v[i];
}

//current node depot, next node first customer,

cnode = 0;
nnode = first;

ttime = d.getDistance(cnode,nnode)/speed;
//travel time from depot to first customer

//find start time for depot, first cust has lowest upper pickup tw

if (first>n/2)
    ctime[0]=r.getLTimeWindow(first)-ttime+r.getServicetime(0);

else {
    float L = r.getLTimeWindow(first+n)
    -d.getDistance(first,first+n)/speed-r.getMaxRideTime();
    ctime[0]= L-ttime+r.getServicetime(0);
}
if(ctime[0]<0)ctime[0]=r.getServicetime(0);

ctime[1] = ctime[0] + ttime + r.getServicetime(nnode);
//time after first cust

//if ctime is lower than lower time window, then wait
if(ctime[1] < r.getLTimeWindow(nnode)+r.getServicetime(nnode)){
    ctime[1] = r.getLTimeWindow(nnode)+r.getServicetime(nnode);
    wtime[1] = ctime[1]-ctime[0]-ttime-r.getServicetime(nnode);
}
//cnode set to first customer
cnode = nnode;

int kkk=1;  //counter in while loop, used in ord and ctime
```

```
///////////////  WHILE LOOP STARTS /////////////////////


//for(int oj=0; oj<2*cun; oj++){//for perhaps better?
while (sumcn > -cn.length || sumcs<sumv ){
    int cek = 0;

    for(int i=0;i<N4.length;i++){N4[i]=-1;}
    for(int i=0;i<N4.length;i++){
    //finds node closest to cnode but not included in N4
    if(kkk<ctime.length)
        nnode=closest(cnode, ctime, N4,kkk);
    else System.out.println("L171 ctime problems");
    //nnode=-2 when no more nodes are left
    if (nnode==-2 && i==0)
        System.out.println("KURT");

    if (nnode == -2){
        break;
    }
    else{  //nnode set into N4
        int count=0;
        for(int j=0;j<N4.length;j++ )
        if(N4[j]!=-1)count++;
        N4[count] =  nnode;

    }//else ends
    }//for ends

    mincost = 3.3E+38f;


    //cost of nodes in N4 evaluated and cheapest chosen
    for(int i=0;i<N4.length;i++){
    if(N4[i] > -1){
        if (mincost > cost(N4[i],cnode,ctime,kkk)){
        mincost = cost(N4[i],cnode,ctime,kkk);
        nnode = N4[i];
        }
    }
    }
    //Here something has gone wrong, algorithm should never have
    //nnode= -2, it should stop before
    if (nnode == -2){
```

```
    System.out.println("L151 BREAK");
    break;
    }//end if

    kkk++;

    visit(nnode);

    ttime = d.getDistance(cnode,nnode)/speed;
    if (kkk<ctime.length){
    ctime[kkk] = ctime[kkk-1] + ttime + r.getServicetime(nnode);

       if(ctime[kkk]<r.getLTimeWindow(nnode)+r.getServicetime(nnode)){
       ctime[kkk]=r.getLTimeWindow(nnode)+r.getServicetime(nnode);
        wtime[kkk]= ctime[kkk]-r.getServicetime(nnode)
        -ctime[kkk-1]-ttime;
    }
    }
    cnode = nnode;

    load[kkk]=load[kkk-1]+r.getLoadChange(cnode);
    //here the while loop has run more times than there are stops ??
    if(kkk>ord.length) {
    System.out.println("NB!!!! EXTRA LOOP, cnode="+cnode);
    cek++;
    }

    //if ok add cnode into route
    if(kkk<ord.length-1 && cnode>0) ord[kkk]=cnode;
    else System.out.println("L289 Error in Route "+cnode);
    if(cek>0)
    System.out.println("Before: sumcs="+sumcs+", sumcn="+sumcn);

    //recalculate stopping criterias
    sumcs=0;
    sumcn=0;

    for(int i=0;i<cs.length;i++){
    sumcs = sumcs + cs[i];
    sumcn = sumcn + cn[i];
    }
    if(cek>0){
    System.out.println("After: sumcs="+sumcs+", sumcn="+sumcn);
    System.out.print(" sumv="+sumv+" cn.length="+cn.length);
    }

}//while ends
```

```
}//route ends


// ------------------- CLOSEST ----------------------- //


//Returns node closest to cnode
 public int closest(int cnode, float[] ctime, int[] N, int k){
 float minldis=3.3E+38f, newldis;
 int cstnode = -2, count;

 //destination of c not in N
 for (int c=0;c<cv.length;c++){
     if(cv[c]!= -1){
     count = 0;
     for (int j=0;j<N.length;j++){
         if(N[j]!=cv[c]+n){count++;}
     }//for ends

     if(count == N.length){
         newldis = separation(cnode, cv[c]+n, ctime, k);

         if(minldis > newldis){
         minldis = newldis;
         cstnode = cv[c]+n;
         }//if ends
     }//if ends
     }//if ends
 }//for ends


 //now it is assumed that the customers travel alone,
 //i.e. demand in each node is 1 or -1
 count = 0;
 for (int c=0;c<cv.length;c++){
     if(cv[c]!= -1) count++;
 }

 if(count == car.getCarCapacity())
     return(cstnode);

 if(count > car.getCarCapacity()) {
     System.out.println("Capacity violated!!!!!!!");
     return(cstnode);
 }
 //origin of c not in N
 for (int c=0;c<cn.length;c++){
      if(cn[c]!= -1){
     count = 0;
     for (int j=0;j<N.length;j++){
         if(N[j]!=cn[c]){count++;}
     }//for ends

     if(count == N.length){
         newldis = separation(cnode,cn[c] ,ctime,k);
         if(minldis > newldis){
         minldis = newldis;
         cstnode = cn[c];
         }//if ends
     }//if ends
      }//if ends
 }//for ends

 return(cstnode);
 }//closest ends

// ---------------------- SEPARATION -------------------- //

//returns space-time separation between node cnode and nnode
public float separation(int cnode, int nnode,float[] ctime,int k){

float ttime, timek1;
float routedur=0, twviol=0,ridetimeviol=0, exride=0, wt=0;


ttime = d.getDistance(cnode,nnode)/speed;
timek1 = ctime[k] + ttime;           //arrival time at nnode

//change in route duration, ctime[k+1]-ctime[k]
routedur = timek1 + r.getServicetime(nnode)-ctime[k];

//tw violations calculated
//for we want to get there soon (twviol>0)
if(timek1>r.getUTimeWindow(nnode))
    twviol=timek1-r.getUTimeWindow(nnode);

//tw viol - if early then increase cost - can wait(twviol<0)
if(timek1<r.getLTimeWindow(nnode))
    twviol=timek1-r.getLTimeWindow(nnode);

if(timek1 < r.getLTimeWindow(nnode)) //if arrival is to early
```

```
    timek1= r.getLTimeWindow(nnode); //wait until ok

//customers ride time violations caluclated,
//if customer has been in the car for longer than max
//ride time sais then ride time viol > 0, service times of
//customer not included, i.e. nnode is a drop off location
//              AND
//customers excess ride times calculated,
//ridetime - direct transport time
if(nnode>n){
    for(int i=1;i<ord.length-2;i++){
    if(ord[i]==nnode-n){
       if(timek1-ctime[i]>r.getMaxRideTime())
        ridetimeviol=timek1-ctime[i];

       exride=
           timek1- ctime[i] - d.getDistance(ord[i],nnode)/speed;

    }//if
    }//for
}//if

//waiting time calculated * persons in the vehicle

int count = 0;
for (int c=0;c<cv.length;c++){
    if(cv[c]!= -1) count++;
}

wt = count*(timek1-(ctime[k] + ttime));

   return(w1*routedur-w2*twviol-w3*ridetimeviol
      -w5*exride+w6*wt+w7*ttime*speed);
}//separation ends



// ------------------------ COST ----------------------- //


//returns cost of visiting nnode
public float cost(int nnode, int cnode, float[]ctime, int k){

int[]vn = new int[4];    //next four nodes considered
int[]N = new int[4];     //empty array used in call to closest
float totcost = 0;
float ttime;             //travel time
```

```
int cc = 0;
float ertime;           //earliest arrival time to node

for(int i=0;i<4;i++){vn[i]=-1;N[i] = -1;}

for(int i=0;i<4;i++){

    totcost = totcost + movecost(cnode, nnode, ctime,k);
    if (nnode>0) visit(nnode);
    vn[i]=nnode;

    ttime = d.getDistance(cnode,nnode)/speed;
    ertime = ctime[k] + ttime;  //arrival time at nnode

    if(ertime<r.getLTimeWindow(nnode))
    ertime=r.getLTimeWindow(nnode);
    cnode = nnode;
    k++;
    ctime[k] = ertime + r.getServicetime(cnode);
    //service at nnode finished
    nnode = closest(cnode,ctime,N,k);
    if (nnode==-2) {
    cc++;
    if (cc==1) {
        nnode = 0;
        totcost = totcost + movecost(cnode, nnode, ctime,k);
    }
    else break;
    }//end if
}//end for

for(int i=vn.length-1;i>=0;i--){
    if(vn[i]> 0)
    unvisit(vn[i]);
}//for ends

return(totcost);
}//cost ends


// ----------------------- MOVECOST ------------------- //


//returns cost of move from current node to next node
public float movecost(int cnode, int nnode,float[]ctime, int k){

float ttime, timek1;
float routedur=0, twviol=0,ridetimeviol=0, exride=0, wt=0;


ttime = d.getDistance(cnode,nnode)/speed;
timek1 = ctime[k] + ttime;           //arrival time at nnode

//change in route duration, ctime[k+1]-ctime[k]
routedur = timek1+wtime[k+1]+r.getServicetime(nnode)-ctime[k];

//tw violations calculated,
if(timek1>r.getUTimeWindow(nnode))
    twviol=timek1-r.getUTimeWindow(nnode);
if(ctime[k] + ttime < r.getLTimeWindow(nnode))
    twviol=r.getLTimeWindow(nnode)-(ctime[k] + ttime);

if(timek1 < r.getLTimeWindow(nnode)) //if arrival is to early
    timek1= r.getLTimeWindow(nnode); //wait until ok

//customers ride time violations caluclated,
//if customer has been in the car for longer than max
//ride time sais then ride time viol > 0, service times of
//customer not included, i.e. nnode is a drop off location
//              AND
//customers excess ride times calculated,
//ridetime - direct transport time
if(nnode>n){
    for(int i=1;i<ord.length-2;i++){
    if(ord[i]==nnode-n){
        if(timek1-ctime[i]>r.getMaxRideTime())
        ridetimeviol=timek1-ctime[i]-r.getMaxRideTime();
        exride=
           timek1- ctime[i] - d.getDistance(ord[i],nnode)/speed;

    }//if
    }//for
}//if
//waiting time calculated * persons in the vehicle
    int count = 0;
for (int c=0;c<cv.length;c++){
    if(cv[c]!= -1) count++;
}

wt = count*(timek1-(ctime[k] + ttime));

return(w1*routedur + w2*twviol + w3*ridetimeviol
       + w5*exride + w6*wt + w7*ttime*speed);
}//movecost ends
```

```
// ----------------------- ORDCOST ------------------- //
//Returns cost of route
public float ordcost(int[] order){

float ttime, routeviol=0, serv, routedur=0,twviol=0;
float rideviol=0,xride=0,wt=0;
int cnode, nnode;

for(int i=0;i<cv.length;i++){cv[i]=-1;cs[i]=-1;cn[i]=v[i];}
load[0]=0;
cnode = order[0];
ctime[0]=retime();
for(int j=0;j<wtime.length;j++)
    wtime[j]=0;

//update ctime, wtime, load for the route found
for(int i=1;i<order.length;i++){
    nnode = order[i];
    load[i]=load[i-1]+r.getLoadChange(order[i]);
    ttime = d.getDistance(cnode,nnode)/speed;
    ctime[i] = ctime[i-1]+ttime+r.getServicetime(nnode)+wtime[i];

    if(ctime[i]<r.getLTimeWindow(nnode)+r.getServicetime(nnode)){
      ctime[i]=r.getLTimeWindow(nnode)+r.getServicetime(nnode);
      wtime[i]=ctime[i]-(ctime[i-1]+ttime+r.getServicetime(nnode));
    }//if ends

    if(i>1 && wtime[i]>0 && load[i-1]>load[i-2]) wait(i);
    //if(diff>-1) totcost = diff;
    cnode = nnode;
    if(cnode>0) visit(cnode);
}//for ends

for(int i=1;i<order.length;i++){
    ttime = d.getDistance(order[i-1],order[i])/speed;
    serv = r.getServicetime(order[i]);

    //tw violations calculated,
    if(ctime[i]-serv>r.getUTimeWindow(order[i]))
      twviol=twviol+(ctime[i]-serv)-r.getUTimeWindow(order[i]);
    if(ctime[i-1]+ttime<r.getLTimeWindow(order[i]))
      twviol=twviol+r.getLTimeWindow(order[i])-(ctime[i-1]+ttime);

    //customers ride time violations caluclated,
    //if customer has been in the car for longer than max
    //ride time sais then ride time viol > 0, service times of
    //customer not included, i.e. nnode is a drop off location
    //              AND
    //customers excess ride times calculated,
    //ridetime - direct transport time
    if(order[i]>n){
    for(int j=1;j<order.length-2;j++){
        if(order[j]==order[i]-n){
        if(ctime[i]-serv-ctime[j]>r.getMaxRideTime()){
            rideviol= rideviol+ ctime[i]-serv-
            ctime[j]-r.getMaxRideTime();
        }

        xride=xride+ctime[i]-serv-ctime[j]
            -d.getDistance(order[i],order[j])/speed;

        }//if
    }//for
    }//if

    //waiting time calculated * persons in the vehicle

    wt = wt+load[i-1]*(ctime[i]-serv-(ctime[i-1]+ttime));

}

if(wt<1E-5)wt=0;

//total route duration
routedur = ctime[ctime.length-1]-ctime[0];

//route duration violations calculated,
//if current time is higer than
//max allowable route duration then route viol becomes > 0

if(ctime[ctime.length-1]-ctime[0]>car.getRouteDuration())
    routeviol =
    ctime[ctime.length-1]-ctime[0]-car.getRouteDuration();


return(w1*routedur+w2*twviol+w3*rideviol
       +w4*routeviol+w5*xride+w6*wt+w7*getDist());

}//ordcost ends


// ------------------------- RETIME -------------------------//
```

```
//Recalculate starting time so that time windows are not violated
public float retime(){
float ttime;

if(ctime[ctime.length-1]>r.getUTimeWindow(ord[ord.length-1]))
    ctime[ctime.length-1]=r.getUTimeWindow(ord[ord.length-1]);

for(int i=ctime.length-1;i>0;i--){
    ttime = d.getDistance(ord[i],ord[i-1])/speed;
    ctime[i-1] = ctime[i]-ttime-r.getServicetime(ord[i]) ;

    if(ctime[i-1]<
       r.getLTimeWindow(ord[i-1])+r.getServicetime(ord[i-1]))
    ctime[i-1]=
        r.getLTimeWindow(ord[i-1])+r.getServicetime(ord[i-1]);

    if(ctime[i-1]>
       r.getUTimeWindow(ord[i-1])+r.getServicetime(ord[i-1]))
    ctime[i-1]=
        r.getUTimeWindow(ord[i-1])+r.getServicetime(ord[i-1]);


}//for ends
return(ctime[0]);
}

// ------------------------- WAIT ----------------------//

//moves waiting times to nodes where there are fewer customers
//in the vehicle
public void wait(int no){
float start=wtime[no];
float diff=0;
for(int i=no;i>1;i--){
    diff=r.getUTimeWindow(ord[i-1])
        -ctime[i-1]+r.getServicetime(ord[i-1]);
    //arrival time within tw
    if(load[i-2]>load[i-1]|| wtime[i]<0.01 || diff<=0)break;
    else{
    if(diff>0 && diff<=wtime[i]){

        wtime[i-1]=wtime[i-1]+diff;
        ctime[i-1]=ctime[i-1]+diff;
        wtime[i]=wtime[i]-diff;
        diff=0;
    }
```

```
     if(diff>0 && diff>wtime[i]){

        wtime[i-1]=wtime[i-1]+wtime[i];
        ctime[i-1]=ctime[i-1]+wtime[i];
        wtime[i]=0;
    }
    }//else ends
}//for ends

}//wait ends


// ------------------------- VISIT -------------------- //


//marks gnode as visited by updating global data
public void visit(int gnode){

for(int c=0;c<v.length;c++){

    if(gnode==v[c]){

    cn[c] = -1;
    cv[c] = v[c];
    break;
    }
    if(gnode==v[c]+n){

    cs[c] = v[c];
    cv[c] = -1;
    break;
    }

}//for ends
}//visit ends


// ------------------------- UNVISIT ------------------- //


//marks gnode as not visited by updating global data
public void unvisit(int gnode){

for(int c=v.length-1;c>=0;c--){

    if(gnode==v[c]){
        cn[c] = v[c];
        cv[c] = -1;

        break;
        }
        if(gnode==v[c]+n){

        cs[c] = -1;
        cv[c] = v[c];
        break;
        }

    }//for ends
    }//uvisit ends

}//class ends


class Request {
    int [][] req;
    float [][] coo;
    Data s;
    int st, max;
    public Request(){}

    //Constructer that initializes matrices and reads in values
    public Request(String filename){
    final int sizereq=4, sizecoo=2;
    Data s = new Data(filename);
    st = s.getStops();
    req = new int[st][sizereq];
    coo = new float[st][sizecoo];
    coo = s.getCoo();
    req = s.getReq();
    max = s.getMaxRideTime();
    }//Request constructer ends

    public int getStops(){return(st);}
    public int getMaxRideTime(){return(max);}

    public float[][] getCooMatrix(){
       return(coo);
   }//getCooMatrix ends

   public float getxCoordinates(int custnum){
       float x= coo[custnum][0];
       return(x);
   }//getxCoordinates ends

    public float getyCoordinates(int custnum){
       float y= coo[custnum][1];
       return(y);
   }//getyCoordinates ends

    public float[] getxy(int custnum){
    float[] xy = new float[2];
    xy[0] = getxCoordinates(custnum);
    xy[1] = getyCoordinates(custnum);
    return(xy);
    }

    public int getServicetime(int custnum){
       int s=req[custnum][0];
       return(s);
   }//getServicetime ends

    public int getLoadChange(int custnum){
       int c=req[custnum][1];
       return(c);
   }//getLoadChange ends

    public int getLTimeWindow(int custnum){
       int tw = req[custnum][2];
       return(tw);
   }//getlTimeWindow ends

    public int getUTimeWindow(int custnum){
       int tu = req[custnum][3];
       return(tu);
   }//getuTimeWindow ends
}//Request class ends

public class Data{
    private LineNumberReader in;
    int[] prob = new int[5];
    float[][]coo;
    int[][]req;
    static int count;
    String filename;
    //method that reads number of depots, number of stops, max duration
    //time of a tour, max capacity of cars, max riding time of customers

    public Data(String file){
    filename = file; readProblem(); read();
```

```
}

public void readProblem() {
try{
    in = new LineNumberReader(new FileReader(filename));
    StringTokenizer dimen;

    //reads the information into a vector prob

    for (int i=1; i<2;i++){
    String dimension = in.readLine();
    dimen = new StringTokenizer(dimension);
    while(dimen.hasMoreTokens()){
        prob[0] = Integer.parseInt(dimen.nextToken());
        prob[1] = Integer.parseInt(dimen.nextToken());
        prob[2] = Integer.parseInt(dimen.nextToken());
        prob[3] = Integer.parseInt(dimen.nextToken());
        prob[4] = Integer.parseInt(dimen.nextToken());
    }//while ends
    }//for ends

}//try ends
catch (EOFException eof) {
        closeFile();
 }
catch (IOException e){
    System.out.println("1 The file "+filename+" could not be opened "
               +e.toString());
    System.exit (1);
}//CATCH ENDS
}//readProblem ends

//method that returnes number of vechicles available
 int getNOCar(){
 return(prob[0]);
}
//method that returnes number of stops
 int getStops(){
 return(prob[1]);
}
//method that returns allowable route duration
 int getRouteDuration(){
 return(prob[2]);
}
//method that returns capacity of cars
 int getCapacity(){
 return(prob[3]);
```

```
}
//method that returns maximum riding time for customers
 int getMaxRideTime(){
 return(prob[4]);
}

//method that reads cooridnates of customers into a matrix coo and
//service time, load change, lower time window and upper time window
//into matrix req

public void read(){
try{
    in = new LineNumberReader(new FileReader(filename));
    StringTokenizer tokens;

    //Reads cooridinates of the customers into a matrix coo
    //and rest into a matrix req

    int dim = getStops()+1;
    int car = getNOCar();
    req = new int[dim][4];
    coo = new float[dim][2];

    int d=0;
    if(dim-1>0 && dim-1<10)    d = 0;
    if(dim-1>9 && dim-1<100)   d = 1;
    if(dim-1>99 && dim-1<1000) d = 2;
    if(car>9 && car<100) d = d +1;

    in.skip(12+d);
    for(int i=0; i< dim+1;i++){
    String tokenstring = in.readLine();
    int index;
    tokens = new StringTokenizer(tokenstring);

    while(tokens.hasMoreTokens()){
        index = Integer.parseInt(tokens.nextToken());
        coo[index][0] = Float.parseFloat(tokens.nextToken());
        coo[index][1]
        = Float.parseFloat(tokens.nextToken());
        req[index][0]
        = Integer.parseInt(tokens.nextToken());
        req[index][1]
        = Integer.parseInt(tokens.nextToken());
        req[index][2]
        = Integer.parseInt(tokens.nextToken());
        req[index][3]
```

```
            = Integer.parseInt(tokens.nextToken());
        }//WHILE ENDS
        } //FOR ENDS

    } //TRY ENDS
    catch (EOFException eof) {
            closeFile();
     }
    catch (IOException e){
        System.out.println("2 The file "+filename+" could not be opened  "
                   +e.toString());
    }//CATCH ENDS

    }//readRequest ends
    private void closeFile() {
        try{
        in.close();
        System.exit(0);
        }
        catch (IOException e) {
        System.err.println("Error closing file" + e.toString());
        System.exit (1);
        }
    }//closeFile ends
    public float[][] getCoo(){return(coo);}
    public int[][] getReq(){return(req);}
} //CLASS Data ENDS

class Distance{
    protected float [][] dist;
    int sto;
    float[][]dcoo;


    public Distance(String filename){
    Request r = new Request(filename);
    sto = r.getStops();
    dist = new float[sto+1][sto+1];
    dcoo=r.getCooMatrix();
    calculateDistance();
    }//Distance Constructur ends

    public void calculateDistance(){
    for(int s=0;s<sto+1;s++){dist[s][s]=0;}
    for(int s1=0;s1<sto+1;s1++){
        for(int s2=0;s2<s1;s2++){
        dist[s1][s2] =
           (float)Math.sqrt((dcoo[s1][0]-dcoo[s2][0])*
                 (dcoo[s1][0]-dcoo[s2][0])+
                 (dcoo[s1][1]-dcoo[s2][1])*
                 (dcoo[s1][1]-dcoo[s2][1]));
        }//for ends
    }//for ends
    for(int s1=0;s1<sto+1;s1++){
        for(int s2=s1+1;s2<sto+1;s2++){
        dist[s1][s2]=dist[s2][s1];
        }//for ends
    }//for ends

    }//calculateDistance ends

    //returnes distance between stop 1 and 2
    public float getDistance(int cust1, int cust2){

    return(dist[cust1][cust2]);
    }//getDistance ends

}//Distance class ends

class Car {
    int capacity, carno;
    String filename;
    Data d;

    public Car(String file){
    filename = file;
    d = new Data(filename);
    }

    public int getCarCapacity(){return d.getCapacity();}
    public int getNOCar(){return d.getNOCar();}
    public int getRouteDuration(){return d.getRouteDuration();}
}//Car class ends

class Chromo extends Request{

    byte[][][]ind;
    int[][] sum;
    int stops, car, pop;

    public Chromo(){}
    public Chromo(int p, int c, int st){
    pop=p; car=c ; stops=st;
    }
```

```
public void createRandInitPop(){
int rand;
ind = new byte[pop][car][stops/2+1];

for(int i=0;i<pop;i++){
    for (int j=0;j<car;j++)
    ind[i][j][0]=1;
    for(int k=1;k<stops/2+1;k++){
        rand=rand_int(car);
        ind[i][rand][k]=1;
    }//for ends
}//for ends
checkPopulation();
//printMatrix(ind[0]);
//printMatrix(ind[1]);
}//createRandIndPop ends

public byte[][][] getPopulation(){return ind;}

public byte[][] getIndividual(int a){
byte[][]parent = new byte[car][stops/2+1];
for(int i=0;i<car;i++){
    for(int j=0;j<stops/2+1;j++){parent[i][j]=ind[a][i][j];}
}

return(parent);
}

public byte[]getCluster(int a, int b){
byte[] clust = new byte[stops/2+1];
for (int i=0;i<stops/2+1;i++){clust[i]=ind[a][b][i];}
return(clust);
}

public void checkPopulation(){
int rand;
for(int i=0;i<pop;i++){
    int sum=0;
    for(int j=0;j<car;j++)
    sum = sum + ind[i][j][0];
    if(sum != car)
    System.out.println("Error for depot "+sum+" car "+car);

    for(int k=1;k<stops/2+1;k++){
    int su=0;
    for(int j=0;j<car;j++){su=su+ind[i][j][k];}
    switch(su){
    case 0:
        rand=rand_int(car);
        ind[i][rand][k]=1;
        break;
    case 1:
        break;
    default:
        System.out.println
        ("Error individiual "+i+" car "+k);
    }//switch ends
    }//for ends
}//for ends

}//checkPopulation ends

public void custInCar(){
sum = new int[pop][car];

for(int i=0;i<pop;i++){
    for (int j=0;j<car;j++){
    for (int k=0;k<stops/2;k++){
        sum[i][j]=sum[i][j]+ind[i][j][k];
    }//for ends
    }//for ends
}//for ends
}//CustInCar ends

public int getCustInCar(int ind,int carno){
custInCar();
return(sum[ind][carno]);
}

//cross over in which there is selected randomly rows from
//both parents, random template created and cross over done
//accordingly.  Offspring is then same as first parent but
//row chosen replaced by cross over line.  Then the offspring
//is checked to see if all customers are included once, if
//not the error is corrected, tried to keep the cross over
//line unchanged, to avoid duplicates: offspring = parent 1

public byte[][] cross_over1(byte[][]inda, byte[][]indb){
int row = inda.length; //# rows
int col = inda[0].length; //# columns in row 0
byte[][]offspring = new byte[row][col];

for(int i=0;i<row;i++){
        for (int j=0;j<col;j++)
        offspring[i][j] = inda[i][j];
    }

    int randa = rand_int(row);
    int randb = rand_int(row);

    int[] template = new int[col];

    for(int i=0;i<col;i++)template[i] = rand_int(2);

    for(int i=0;i<col;i++){
        if(template[i]==1){offspring[randa][i] = indb[randb][i];}
    }//for ends
    offspring = correct_Matrix(offspring, randa);
    return(offspring);
    }//cross1 ends


    public byte[][] cross_over2(byte[][]inda, byte[][]indb, float just){
    int row = inda.length; //# rows
    int col = inda[0].length; //# columns in row 0
    int rand;
    byte[][]offspring = new byte[row][col];

    for(int i=0;i<row;i++){
        for (int j=0;j<col;j++)
        offspring[i][j] = inda[i][j];
    }

    //choose a random row from both parents
    int randa = rand_int(row);
    int randb = rand_int(row);

    //recipe of row randa in offspring, if value of template[i]=1
    //then offspring[randa][i]=indb[randb][i], if template[i]=0
    //then offspring[randa][i]=inda[randa][i]

    int[] template = new int[col];

    for(int i=0;i<col;i++){
        rand = rand_int(100);
        if (rand>just-1)
        template[i] = 1;
    }//for ends
    for(int i=0;i<col;i++){
        if(template[i]==1) offspring[randa][i] = indb[randb][i];
```

```
}
offspring = correct_Matrix(offspring, randa);
return(offspring);
}//cross2 ends

public void printMatrix(byte[][] matrix){
  int row = matrix.length, col=matrix[0].length;
  System.out.println();
  for(int i=0;i<row;i++){
    for(int j=0;j<col;j++){System.out.print(matrix[i][j]);}
    System.out.println();
  }
}

//random customer chosen and moved to another car
public void mutation (byte[][] matrix){
int i=0,out=0,row = matrix.length, col=matrix[0].length;
int randcust = rand_int(col);
while(out==0){out=matrix[i][randcust];i++;}
matrix[i-1][randcust]=0;
int rand = rand_int(row);
while (rand==i-1){rand=rand_int(row);}
matrix[rand][randcust]=1;
}

//returns a random integer on the interval [0, L-1]
public int rand_int(int L){ return((int)(Math.random()*L));}

public byte[][] correct_Matrix(byte[][] matrix, int randa){
int rand, row = matrix.length, col=matrix[0].length;
for(int j=0;j<row;j++){matrix[j][0]=1;}

for(int i=1;i<col;i++){
   int su=0;
   for(int j=0;j<row;j++){su=su+matrix[j][i];}
   switch(su){

     case 0:
       rand=rand_int(row);
       while (rand == randa) {rand=rand_int(row);}
       matrix[rand][i]=1;
       break;
     case 1:
       break;
     case 2:
     for(int j=0;j<row;j++){
         if (matrix[j][i]==1 && j!=randa) matrix[j][i]=0;
         }
           break;
         default:
           System.out.println("Something has gone wrong in cross_over");
       }//switch ends
    }//for ends
    return(matrix);
    }//correct_Matrix ends

    public int getUTimeWindow(int c){
    return(super.getUTimeWindow(c));
   }
}//class ends
```