# Transport Optimization at Danisco: Routing of Pack Machines

Sara Bisander Nielsen

Kongens Lyngby 2006
IMM-Master's Thesis-2006-46

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

# Abstract

The main purpose of the thesis is to solve the two real-life distribution problems, that is the pack machine problem and the limited pack machine problem. Well known problems from operational research are used as approaches to formulate the problems, these are the Vehicle Routing Problem, the Vehicle Routing Problem with Time Windows and the Travelling Salesman Problem (with Time Windows). Mathematical models of the approaches are presented, and it is described how they could be applied on the pack machine problem and the limited pack machine problem.

Solution methods based on the local search method Steepest Descent and the metaheuristic Simulated Annealing are developed and implemented. Initial solutions are obtained with a nearest neighbour construction heuristic and an insertion heuristic. Results obtained with Simulated Annealing and Steepest Descent is presented.

Finally an exact method for solving the pack machine problem is presented. The method is based on column generation and branch & bound.

Keywords: *Distribution Problems, Vehicle Routing Problem, Time Windows, Simulated Annealing, Steepest Descent.*

# Resumé

Det primære mål med projektet er at løse to konkrete distributions problemer, pakkemaskine problemet og det begrænsede pakkemaskine problem. Problemer kendt fra operationsanalyse benyttes som fremgangsmåde til at formulere problemerne. Disse problemer er Vehicle Routing Problemet, Vehicle Routing Problemet med tidsvinduer og Travelling Salesman problemet (med tidsvinduer). Matematiske modeller af disse fremgangsmåder er præsenteret, og det er beskrevet, hvordan de kan anvendes på henholdsvis pakkemaskine problemet og det begrænsede pakkemaskine problem.

Løsningsmetoder baseret på den lokale søge metode *bedste trinvise forbedring* og metaheuristiken *simuleret udglødning* er udviklet og implementeret. Initielle løsninger er fundet med konstruktions heuristikken *nærmeste nabo* og en indsættelses heuristik. Resultater opnået med simuleret udglødning og bedste trinvise forbedring præsenteres.

Til sidst er en eksakt metode til at løse pakkemaskine problemet præsenteret. Metoden er baseret på *søjle generering* og *Branch & Bound*.

Nøgleord: *Distributions problemer, Vehicle Routing problemet, tidsvinduer, simuleret udglødning, bedste trinvise forbedring.*

# Preface

This master's thesis was carried out at the Department of Informatics and Mathematical Modelling, at the Technical University of Denmark. The thesis was carried out in co-operation with the Danish Sugar Firm Danisco. Jesper Larsen and Jens Clausen, IMM•DTU and Allan Larsen CTT•DTU were supervisors on the project.

I would like to thank my supervisors for excellent supervision, and Anders Hebsgaard at Danisco for providing the neccesary informations and answering my questions immediately. Furthermore I would like to thank Kirstine Thomsen and Thomas Bang Nielsen for commenting my work. Lyngby, May 2006

Sara Bisander Nielsen

# Contents

Chapter 1

# Introduction

## 1.1 Motivation

In the distribution industry, great retrenchment could often be gained by using *Operational Research*. The company Danisco was aware of this, and made an application to DTU, about a distribution project. I saw possibilities in this project, and decided to use the problem for my master's thesis.

The problem which should be solved is a real-life routing problem. It is provided by the danish sugar producer Danisco. The task in this project is to minimize the cost of the given problem. The cost is defined as the distribution cost. The economic issues in developing a routing strategy will always be important, and that is why this problem and the solution will be relevant for Danisco, both now and in the future.

The unemployment is at the moment historically low, so the issue of reducing the man hours in the scheduling phase has great relevance at the moment. Furthermore, the issue of scheduling the routes manually is quite cumbersome. The scheduling requires one working week for an employer according to Danisco. By using methods known from *Operational Research* this could be done in much less time, and by a computer which is cheaper in salary than humans. It should be noted, that the reduction of work hours for scheduling the plans is a side effect, and not on focus in this project.

The problem treated in this project is not an outstanding problem for Danisco. There are similar problems in the other countries, where Danisco has subsidiaries. It is a part of the aim of the project, that the methods, which are developed to solve this particular problem should be convertable such that they could be applied to the other problems. An investigation of their ability to solve such problems is however not carried out.

The routes are until now planned manually. The customers remain more or less the same each year, hence the scheduling only requires one week. The schedule from one year can be used as a base to the next years schedule, by incorporating the minor changes. In the coming years however, the schedule from the last year can not be reused, since the problem has changed considerable. The reason for this change is described in the next paragraph.

The EU Commissioner for Agriculture and Rural Development made in 2005 a reform concerning the production of sugar. The objective is to obtain a European sector of sugar, which is competitive at a global level. Some of the actions from the reform, were to lower the guaranteed price on sugar with 64% and to restrict the national sugar quotas further [1]. These restrictions has a great influence on Danico's sugar production and future strategies. Among the changes are the close downs of the factory in Assens, Denmark and the factory in Köpingebro, Sweden after the *campaign* in 2006 [2]. How these changes affect this problem, is described in Section 1.6.

## 1.2 Problem Description

The problem described in this section, is the original problem, and it does not consider the prospect of the close down of factories. The problem is called the *pack machine problem*.

When Danisco produces sugar, a by-product which can be used as feed, is also produced. This by-product is delivered to farms as *bulk*. The feed is packed by a pack machine in enormous bags to improve the keeping quality. The feed can not be transported, when it is packed, so it should be packed at the farms, which are referred to as the *customers*. Trucks are transporting the feed from the factory to the customers, the pack machine arrives either at the same time or immediately after. The pack machines are the limiting factor, since these are transported by a tractor, and hence is transported at a low speed. The customers are visited either one or two times during the *campaign* which is the period, where sugar and feed is produced. The customers have the opportunity to give a *time window*, which is a period, where they want to be serviced. Danisco strives to fulfill these time windows, but no guarantee is given.

The overall objective is to minimize the total amount of time used by the pack machines.

Some technical specifications and statistics for the problem are listed here:

- There are four pack machines, three of these are located in Sweden, while the last one is located in Denmark. In this project, only the Swedish pack machines are considered.
- There are two depots for the pack machines: one in Sweden and one in Denmark.
- Two of the Swedish pack machines produce 50 tons of bulk per hour, whereas the last one produces 35 tons of bulk per hour.
- The customers can choose whether they should be serviced in the first or the last part of the campaign. This gives reason for some *wide* time windows.
- There are between 500 and 700 customers in Sweden. There are 570 customers in the dataset provided by Danisco, and used in this project.
- The campaign length, i.e. the time period where bulk is produced, is 90 days.
- The pack machine will only return to the depot during the campaign if the machine breaks down - this case will not be considered in the project.
- The Danish customers are visited once during the campaign.
- The Swedish customers near the factories are visited twice during the campaign, while the customers located further away are visited once. The visits are described as *double visits*, if they belong to the same customer.
- Due to weather conditions, customers in the northern part of Sweden are serviced first.
- On average 400 tons of bulk is packed each day.
- Two of the Swedish pack machines are working around 14 hours per day, while the last one is working around 10 hours per day.
- The pack machine can be transported from one customer to another at highest 30 km/h, since this is the speed limit for tractors in Sweden.

This problem description, gives reason to some considerations. These will be presented in Section 1.4.

## 1.3 Contribution

The contribution of this thesis is to clarify and formulate the two problems provided by Danisco, that is the pack machine problem and the limited pack machine problem, and solve the problems. Solution methods are developed and implemented to solve the pack machine problem, and furthermore tested on the limited pack machine problem.

The requirements of the solution methods are, that they should be effective, such that good but not neccesary optimal solutions are obtained fast. The solution methods should be described thoroughly, such that the implementation could be improved further by Danisco.

The results should clarify which approach is the most adequate, when solving the pack machine problem and the limited pack machine problem.

Another goal is to develop an exact method for the pack machine problem. The method is not implemented, as it is supposed, that this method does not fulfill the requirements of obtaining good solutions fast.

## 1.4 Considerations

The considerations about the problem formulation concern the cost function and the working time versus the transportation time, as presented in Section 1.4.1 and 1.4.2.

### 1.4.1 Cost Function

The cost function could be a subject of consideration, as there are other objectives than the total amount of time, which could be optimized. Some other possible objectives are presented in the next paragraphs.

**Total Distance** Minimizing the total distance could be another objective, this objective is compatible with the goal of minimizing the amount of time used, as a short total distance will reduce the amount of time used. Some experiments have been performed, where minimization of distance is the objective.

**Pack Machines** Minimizing the number of used pack machine is another relevant objective, if a solution could be obtained with only two pack machines for the pack machine problem, it would be preferable, as these machines are expensive.

**Different Salary** If the personnel which operates the pack machine demands different salary according to the working time, it would be an objective to minimize the salary. Usually personnel gets higher salary in the evening and in the weekends.

**United Waiting Time** If it is possible to create longer periods with continuous waiting time for a pack machine, it might be possible to service for example Danish customers with the pack machine. Including the Danish customers in the model, would gain the same effect.

### 1.4.2 Working Time versus Transportation Time

It is also a subject of consideration if the transportation time should be included in the 10 and 14 hours the pack machines are working each day. If the transportation time is not included, the amount of days used would be less, but the problem would be more complex, since different types of time periods should be modelled, such as a transportation period, a packing and transportation period and a rest period. As it turns out the transportation time is included in the working hours. It therefore seems reasonable, that the working time for the pack machines can be treated as it was one connected period of $90 \cdot 14$ or $90 \cdot 10$ hours, and not 90 distinct periods. But you have to be careful, since the time windows are given with respect to day length which is 24 hours. How this is treated, will be discussed later in Section 2.2.

## 1.5 Limitations and Assumptions

In this section, the limitations and the assumptions for the problem are described. In general, the formulated and solved problem should be as realistic as possible. But solving a full realistic problem is beyond the scope for this project. In the following the limitations are presented.

**Static problem.** The problem is static, such that only customers, which are present before the beginning of the campaign, are considered. According to Danisco, the number of customers which arrives after the start of the campaign length is limited. The same is valid for customers which cancel during the period. The program could however rather easily be improved to take hand of a dynamic problem.

**Euclidean distances.** The distances between customers are euclidean. The distance is therefore shorter than the actual road distances. This limitation is made, because a shortest path calculation on the Swedish road system, would be inconvenient compared to the results gained. The limitation will in most cases not affect the solution much. But especially around the island *Øland*, there might occur some unrealistic solutions. The method presented in this project, would however work in the same way, when road distances are available.

**Pack machines.** Only the routing problem of the pack machines is considered in this project. Another routing problem, could be based on the trucks, which supply the customers with bulk. The trucks have a higher maximum speed and they have to return to the factory often, in order to provide the bulk, which has limited keeping quality, to the customers. When building a realistic model, the problem with the trucks should be taken into consideration. Not as an independent problem, but as a part of the routing problem for the pack machines, where constraints guarantee that bulk is provided to the customers when the pack machine arrives. The reason for not involving this routing problem in the project, is to keep focus on the pack machine problem.

**Only Swedish customers.** The problem consists only of Swedish customers. It would however be more appropriate to include the Danish customers too. The strengthening of the Sound region[1], makes it easier to coorporate between country borders. It seems obvious, that a better overall solution can be obtained,

[1] In Danish: Øresundsregionen

if the Danish customers, and the pack machine servicing these, are included in the model.

**Unlimited keeping quality.** It is assumed, that the bulk, which is packed has unlimited keeping quality. This is however, far from realistic, since the bulk has very limited keeping quality, before it is packed by the pack machines. The bulk is produced continuously, and should be provided to the customers as soon, as it is produced. This affects the problem in the way, that the pack machines should service customers instantly. Such that it is not allowed, that all pack machines are transported through long distances simultaneously. On the other hand, it is not possible to provide more bulk occasionally. This leads to a problem, where the ratio between transportation and packing should be equally distributed during the campaign. This is however beyond the limits of this project.

**No personal considerations.** The only focus in this project is the economic issue. That is, a good solution, is a solution, which minimizes the total amount of work hours. This might give some inexpedient solutions, seen from the drivers point of view. It is for example not guaranteed, that the driver are neither in nor near a town at closing time.

**No break downs.** It is assumed, that neither the factories nor the pack machines will break down, such that no back up plans are made.

## 1.6 The Limited Pack Machine Problem

The close down of the factories in Assens and Köpingebro will result in a smaller sugar and feed production. As a result of this, only two pack machines are available in Sweden. The wide time windows consisting of either the first or the last part of the campaign, are still valid. The new problem is called the *limited pack machine problem*. The problem is new for Danisco, hence no valid solution is provided by Danisco for the limited pack machine problem.

## 1.7 Outline of the Thesis

The rest of the thesis is organized in the following way:

**Chapter 2** Two different approaches for the pack machine problem are presented. The two approaches are formulated as mathematical models based on two well known optimization problems. A mathematical model for the limited pack machine problem is also presented. Some theory concerning the complexity of optimization problems is presented, and finally the models are compared, while a new approach for the double visits is presented.

**Chapter 3** The literature used in this project is reviewed. The literature reviewed concerns the three problems which are used as basis for modelling the pack machine and the limited pack machine problem.

**Chapter 4** The information given in the data set provided by Danisco is presented. The original solution to the pack machine problem is presented, the solution was provided by Danisco.

**Chapter 5** Solution methods for the two approaches used to model the pack machine problem is presented. Finally it is described how the methods for these approaches could be used to solve the limited pack machine problem.

**Chapter 6** The structure of the programs is illustrated, and the implementation is described.

**Chapter 7** The values of the parameters for some of the solution methods are tuned. It is described how the parameter tuning is performed, and the results of the parameter tuning are presented.

**Chapter 8** The results obtained by the solution methods are presented as follows: First the results of the two approaches for the pack machine are presented and discussed briefly, finally the results for the limited pack machine problem are presented.

**Chapter 9** An exact method for solving the pack machine problem is presented. The method is not implemented, but is included in this thesis for academic reasons. A method for finding a lower bounds for the problem is presented as a part of the exact method. It is also presented how these lower bounds could be improved.

**Chapter 10** An overview of what have been achieved in this thesis is presented and discussed. Some considerations concerning conflicting objectives in the problem formulation and the relevance of this thesis for Danisco is presented. Finally it is presented how the solutions could be improved, and how the problem could be extended.

**Chapter 11** The conclusions of the thesis is presented. The conclusion is followed by the references in the bibliography.

**Appendix A** A map of Sweden including the different geographic zones provided by Danisco is presented. The data set provided by Danisco is presented.

**Appendix B** Some additional results are presented, i.e. results not included in Chapter 7 and 8.

**Appendix C** The source code for the programs is presented.

Chapter 2

# Formulation of Mathematical Models

The overall aim for this project is to solve the *Pack machine Problem*. It is a practical problem, and it can not be described precisely by some of the already known problem types. Some of the well known and studied optimization problems can constitute a basis when modelling the problem. There are several approaches to how the model can be formulated, which results in different types of models. In Section 2.1 and 2.2 two different approaches and the resulting models are presented, respectively models based on the *Vehicle Routing Problem (VRP)* and the *Vehicle Routing Problem with Time Windows (VRPTW)*.

The model described in Section 2.3 considers the problem presented in Section 1.6.

In Section 2.4 the theory necessary for understanding the complexity of the models is presented. In Section 2.5 the different models are compared.

All models are defined on a complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where the vertices $\mathcal{V}$ are the customers and the depot. There are $n$ customers. The start depots are denoted $d_k^+$, the depots could be some of the customers, but is instead the real depot for the pack machines. These depots are identical, for each of the vehicles, but are modelled as different depots in this model. The arcs $\mathcal{A}$ connects the vertices. $\mathcal{K}$ is the set of pack machines, there are $m$ pack machines. The decision variable $x_{ij}$ is a binary variable, which equals 1 if the arc $(ij)$ is used in the solution and 0 otherwise. $c_{ij}$ is the transportation cost for driving from customer $i$ to $j$.

## 2.1 The VRP Model

One approach to solve this problem, is to send all vehicles to the northern part Sweden, which is where the customers need to be serviced at the beginning of the campaign period. Once all customers in the northern part of Sweden are visited, the customers in the central and southern part of Sweden are serviced. This approach will divide the problem in two distinct problems. The two problems are described in the next two paragraphs.

**The problem for the customers in northern Sweden.** These customers are treated separately, as they should be serviced first. The problem is a Vehicle Routing Problem. The routes should start at the depot, but they can end at an arbitrary customer.

**The problem for the customers in central and southern Sweden.** This is a multi depot VRP, i.e. a problem with a different start- and end-depot.

There exists many variants of the Vehicle Routing Problem, the basis problem is known as the Capacitated Vehicle Routing Problem (CVRP), and it can be described as follows:

A set of customers should all be serviced once by a vehicle. All vehicles start from a depot, and after servicing a number of customers they return to the depot. The customers have demands (given amounts of goods), which should be fulfilled. The vehicles have capacities on how much goods, they can be loaded with. The objective is to find a set of routes fulfilling the demands, where either the total transportation cost should be minimized or the number of vehicles used should be minimized. The two objectives can also be combined in a weighted cost function.

The model for the Capacitated Vehicle Routing Problem can be used as a basis for the model of the pack machine problem, where some modifications must be added. There are for example no limitations on the capacities, since the customers do not have demands on goods, but on working hours. There is also the restriction, that some customers should be serviced first, due to the weather conditions. Furthermore there is a strict limit on the number of vehicles.

The customers are divided in two sets, one for the customers in the northern part of Sweden and one for the customers in the central and southern part of Sweden. The set of customers in the northern part of Sweden are called $\mathcal{V}_{north}$. The model based on the Vehicle Routing Problem, concerning the customers in the northern part of Sweden, can be formulated mathematically in the following way, when minimizing the total transportation cost:

$$\min \sum_{i \in \mathcal{V}_{north}} \sum_{j \in \mathcal{V}_{north}} c_{ij} x_{ij} \tag{2.1}$$

subject to

$$\sum_{i \in \mathcal{V}_{north}} x_{ij} = 1 \quad \forall \quad j \in \mathcal{N} \setminus \{0\} \tag{2.2}$$

$$\sum_{j \in \mathcal{V}_{north}} x_{ij} = 1 \quad \forall \quad i \in \mathcal{N} \setminus \{0\} \tag{2.3}$$

$$\sum_{i \in \mathcal{V}_{north}} x_{id_k^+} = 1 \quad \forall \quad k \in \mathcal{K} \tag{2.4}$$

$$\sum_{i \notin \mathcal{S}} \sum_{j \in \mathcal{S}} x_{ij} \geq r(\mathcal{S}) \tag{2.5}$$

$$x_{ij} \in \{0, 1\} \quad \forall \quad i, j \in \mathcal{V} \tag{2.6}$$

The constraints given in (2.2) and (2.3) guarantee, that exactly one vehicle will arrive at each vertex and exactly one vehicle will depart from each vertex, not including the depot. The constraint in (2.4) imposes every vehicles $k \in \mathcal{K}$ to depart from the depot. The constraint in (2.5) can be interpreted in the following way: The set $\mathcal{S}$ is a subset of $\mathcal{V}$, for which it holds that $\mathcal{S} \neq \emptyset$ and $\mathcal{S} \subseteq \mathcal{V}$. $r(\mathcal{S})$ is the minimum number of vehicles needed to service the customers in $\mathcal{S}$, this constraint is also called the *capacity cut constraint.* As mentioned before, the decision variable should be binary, as (2.6) guarantees.

The model could be applied to the customers located in the central and southern part of Sweden, by letting the southernmost customer in each of the routes in the northern part of Sweden be the start depot $d_k^+$ for each of the vehicles. The actual depot in Scania, will be the end depot $d_k^-$ for all vehicles. A customer which should be serviced twice, is modelled as two customers. The first customer has demand as the first visit, and the second customer has demand as the second visit. The coordinates for the two new customers are the same as for the original customer.

This problem has a great disadvantage, when dealing with the customers, which should be serviced twice. In an optimal solution, these visits will be scheduled immediately after each other, and will then appear as just one bigger visit. This is described further in Section 8.1 and Appendix B.2. This is indeed not as intended in the pack machine problem. The model described in Section 2.2 uses an approach with time windows to handle this problem.

## 2.2 VRPTW Model

Time windows could be used to separate the double visits. Furthermore they could be used to secure that customers in the northern part of Sweden are serviced first. The time windows are constructed from the solution provided by Danisco, such that the customers should be serviced at +-5 days from the delivery date according to the schedule provided by Danisco.

In this section the model from Section 2.1 is extended to include time windows. A time window is a period where the customer should be serviced. the time windows are often specified by the customers. In this problem, the time windows are given as the delivery date according to the solution provided by Danisco + / - 5 days. These days consist of 24 hours, while the working days consist of 10 or 14 hours. Hence one working day equals 10/24 or 14/24 normal days.

The model presented is a variant of the general VRPTW. The formulation is a multi commodity network flow model with time windows as described in [3].

In the VRPTW model, two decision variables are defined, where the first corresponds to $x_{ij}$ in the VRP model, with an extra index indicating which pack machine should be used. The defined decision variables are:

$x_{ijk}$ is a binary variable, which is 1 if pack machine $k$ drives directly from customer $i$ to $j$, and 0 otherwise.

$w_{ik}$ indicates the arrival time for pack machine $k$ at customer $i$.

The following parameters are defined:

$c_{ij}$ is the transportation cost for driving from customer $i$ to $j$. The transportation for driving from $j$ to $i$ is the same, and hence the problem is symmetric.

$p_i$ is the amount of bulk, which should be delivered at customer $i$.

$f_k$ is the time, machine $k$ uses to produce one unit of bulk.

$s_{ik}$ is the service time if customer $i$ is serviced by machine $k$, note that $s_{ik} = f_k * p_i$.

$t_{ij}$ is the travel time from customer $i$ to $j$.

$a_i$ is the earliest arrival time to customer $i$.

$b_i$ is the latest arrival time to customer $i$.

$E$ is the earliest departure time from the depot.

$L$ is the latest arrival time to the depot.

The reason for both having the travel time $t_{ij}$ and the transportation cost is, that these are not neccesary identical. The cost of travelling could be weighted compared to the service time on the objective function in (2.7).

The problem can be formulated mathematically as:

$$\min \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} (c_{ij} + s_{ik}) x_{ijk} \tag{2.7}$$

subject to

$$\sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{N}} x_{ijk} = 1 \quad \forall \quad i \in \mathcal{V} \tag{2.8}$$

$$\sum_{j \in \mathcal{N}} x_{d_k^+ jk} = 1 \quad \forall \quad k \in \mathcal{K} \tag{2.9}$$

$$\sum_{i \in \mathcal{N}} x_{ijk} - \sum_{i \in \mathcal{N}} x_{ijk} = 0 \quad \forall \quad k \in \mathcal{K}, j \in \mathcal{N} \tag{2.10}$$

$$\sum_{i \in \mathcal{N}} x_{i,n+1,k} = 1 \quad \forall \quad k \in \mathcal{K} \tag{2.11}$$

$$x_{ijk}(w_{ik} + s_{ik} + t_{ij} - w_{jk}) \leq 0 \quad \forall \quad k \in \mathcal{K}, (i,j) \in \mathcal{A} \tag{2.12}$$

$$a_i \sum_{j \in \mathcal{N}} x_{ijk} \leq w_{ik} \leq b_i \sum_{j \in \mathcal{N}} x_{ijk} \quad \forall \quad k \in \mathcal{K}, i \in \mathcal{N} \tag{2.13}$$

$$E \leq w_{ik} \leq L \quad \forall \quad k \in \mathcal{K}, i \in \{d_k^+, d_k^-\} \tag{2.14}$$

$$w_{ik} \geq 0 \quad \forall \quad k \in \mathcal{K}, i \in \mathcal{V} \tag{2.15}$$

$$x_{ijk} \in \{0, 1\} \tag{2.16}$$

The objective function defined in (2.7) states the overall time used by the pack machines, i.e. the complete transportation and working time for the machine. Another objective could be to minimize the total distance of the route, in this case the objective function would be as described in Equation (2.17).

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij} \sum_{k\in\mathcal{K}} x_{ijk} \tag{2.17}$$

Constraint (2.8) guarantees that each customer is assigned to exactly one pack machine. The constraints (2.9) and (2.11) guarantee that every pack machine leaves and arrives at a depot exactly once. Constraint (2.10) guarantees, that the number of machines that arrives at customer $j$ also leaves customer $j$. Constraint (2.12) can be described in the following way. If $x_{ijk} = 1$, then the following should be fulfilled: $w_{ik} + s_{ik} + t_{ij} \leq w_{jk}$, that is, if machine $k$ drives directly from customer $i$ to customer $j$, then the earliest service time for customer $j$ is when machine $k$ has serviced customer $i$ and has travelled from customer $i$ to $j$. Constraint (2.13) can be interpreted in the following way: If machine $k$ drives directly from $i$ to $j$ ($\sum_{j\in\Delta^+(i)} x_{ijk} = 1$), then the arrival at customer $i$ ($w_{ik}$) should lie in the time window $[a_i, b_i]$. Constraint (2.14) guarantees that the time windows of the depots are fulfilled. Equation (2.15) is a non negativity constraint. The decision variable $x_{ijk}$ should be binary, as constraint (2.16) guarantees.

## 2.3 Double TSP Model

The problem which arises after closing down one factory, and reducing the available pack machines to two is called the *limited pack machine problem*. This problem could still be solved as a VRPTW problem with two pack machines and no narrow time windows at +/- 5 days from the original delivery date. The time windows which divide the campaign in two periods are still valid. The employer, who manage the route schedules, proposed another approach to this problem, where the customers were divided in two groups, each belonging to one pack machine. Since most of the customers are located in Scania, these will constitute one group. The customers located in the rest of Sweden will constitute the other group.

The customers can choose to be serviced either in the first or in the second part of the campaign. This leads to two open Travelling Salesman Problems, Which should be connected such that, the total length of the two tours are minimized, and not the length of the tours separately. One way of modelling this, could be with time windows, such that the time windows corresponds to either the first or the second part of the campaign. This approach could give some problems according to the customers which are serviced twice. This is described further in Section 2.3.2.

The Travelling Salesman Problem is one of the most well studied optimization problems. It can be described in the following way: One salesman has to visit $n$ customers or cities exactly once. The total distance travelled by the salesman should be minimized. The problem is defined on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of vertices representing the cities the salesman has to visit, and $\mathcal{A}$ is the set of arcs, the travelling salesman can use. The graph would often be complete, which is also the case for the two problems described in this section. The defined decision variables are:

$x_{ij}$, a binary variable, which is 1 if the salesman travels directly from city $i$ to city $j$ and 0 otherwise.

The parameters used in the model are:

$c_{ij}$, the transportation cost for travelling directly from customer $i$ to $j$.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.18}$$

subject to

$$\sum_{j=1:j \neq i}^{n} x_{ij} = 1 \quad \forall \quad i \in \mathcal{V} \tag{2.19}$$

$$\sum_{i=1:i \neq i}^{n} x_{ij} = 1 \quad \forall \quad j \in \mathcal{V} \tag{2.20}$$

$$\sum_{i \in \mathcal{S}} \sum_{j \notin \mathcal{S}} x_{ij} \geq 1 \quad \forall \quad \mathcal{S} \subset \mathcal{V}, \mathcal{S} \neq \emptyset \tag{2.21}$$

The objective function is defined in (2.18), and it states the complete transportation time for the salesman. The constraints in (2.19) and (2.20) guarantee that the salesman leaves and arrives at each city exactly once. These constraints are the same as in the *assignment problem*. The constraint in (2.21) is a *cut-set* constraint. It guarantees that the tour travelled by the salesman is connected, that is, all *cuts* described in (2.21) has a capacity which is greater than one. The Travelling Salesman Problem is $\mathcal{NP}$-hard, so there exists no algorithms, which can solve the problem to optimality in polynomial time. In the next section, it is described, how the TSP can be applied to the limited pack machine problem.

### 2.3.1 The Limited Pack Machine Problem Modelled as TSP

The TSP-model can be applied to the limited pack machine problem, if the constraints (2.22) and (2.23) are added to the formulation. The variable $w_i$ corresponds to $w_{ik}$ in the VRPTW formulation, and states the arrival time at customer $i$. The parameter $s_i$ states the service time at customer $i$, $t_{ij}$ the travel time between customer $i$ and $j$ and $[a_i, b_i]$ gives the time window for customer $i$.

$$x_{ij}(w_i + s_i + t_{ij} - w_j) \leq 0 \quad \forall \quad (i,j) \in \mathcal{A} \tag{2.22}$$

$$a_i \sum_{j \in \mathcal{N}} x_{ij} \leq w_i \leq b_i \sum_{j \in \mathcal{N}} x_{ij} \quad \forall \quad i \in \mathcal{V} \tag{2.23}$$

The constraint given in (2.22) manages the time as constraint (2.12) in the VRPTW formulation. The constraint given in (2.23) guarantees, that the time windows are fulfilled.

An approach of dividing the customers in sets according to the wide time windows and solving the problems as TSPs could also be used. In this approach it is important, that the distance between the last customer in the first period and the first customer in the last period, should be located not too far away, as this could cause some bad solutions.

### 2.3.2 The Problem in Scania

All customers which has double visits are located in Scania. It is assumed, that customers which have double visits, has one visit in each of the periods. Even though it is not guaranteed, that the solution to this problem is sensible, regarding the double visits, as some of the double visits might be scheduled close, if the first visit is scheduled in the last part of the first period, and the second in the first part of the last period. The problem concerning the double visits will be described further in Section 2.5. Danisco has proposed the following approach for handling the double visits. The customers in Scania are divided into two sets of customers. The first set of customers are those, who should be serviced in the first part of the campaign. The second set contains customers, which should be serviced in the second part of the campaign. This leads to two TSPs. The first TSP is solved either to optimality or by a heuristic, without paying attention to which of the customers should be serviced twice. The second route is constructed such that customers with double visits are visited in the same order as in the first route. The remaining customers are inserted in between these customers, such that the route is minimized.

## 2.4 Complexity

As described before the pack machine problem can be formulated as a type of VRP or VRPTW, and the limited pack machine problem can be formulated as a TSP. All of these problems are among the most difficult optimization problems to solve. The theory behind how to classify a problem as a difficult problem is known as complexity theory, and it will be described briefly in this section. The description in [4], is used as source in this description. Firstly the following types of problems are introduced.

**An Optimization Problem** is a problem, where the optimal solution should be found according to some constraints. The solution is evaluated by a cost function, and is solved when the cost function has reached its optimal value.

**A Decision Problem** is a problem where the answer is either *yes* or *no*. An example of a decision problem, is if there exists a solution to an optimization problem where the cost function is highest a given value or not. This problem is made from an optimization problem. And it is always possible to make a decision problem from an optimization problem. An example of an optimization problem is $\min\{cx : x \in \mathcal{S}\}$. A corresponding decision problem is: does there exists an $x \in \mathcal{S}$ where $cx \leq k$?

The following complexity classes can be defined for a decision problem.

$\mathcal{NP}$ is the class of decision problems, where, given the answer to the problem is *yes*, it can be proved in polynomial time.

$\mathcal{P}$ is the class of decision problems, which can be solved in polynomial time.

It is not known whether $\mathcal{NP} = \mathcal{P}$ or $\mathcal{NP} \neq \mathcal{P}$. If $\mathcal{NP} \neq \mathcal{P}$, there exists a class

$\mathcal{NP} \setminus \mathcal{P} \neq \emptyset$. Decision problems belonging to this class, can not be solved in polynomial time.

Another class, which is a subset of $\mathcal{NP}$ is defined in the next paragraph.

$\mathcal{NPC}$, the class $\mathcal{NPC}$ ($\mathcal{NP}$-complete), is a special class of problems. It is the case for all problems in $\mathcal{NP}$, that they can be converted to problems in $\mathcal{NPC}$ in polynomial time.

If a problem $P$ can be converted to another problem $Q$ in polynomial time, the problem $P$ is said to be *polynomial reducible* to $Q$. If $P$ is polynomial reducible to $Q$ and $Q$ can be solved in polynomial time, then $P$ can also be solved in polynomial time. On the other hand, if $P$ cannot be solved in polynomial time, then neither can $Q$.

For optimization problems the following class is defined.

$\mathcal{NP}$-hard, is the class of optimization problems, where the decision problems belonging to them are $\mathcal{NP}$-complete.

## 2.5 Comparing the Models

In Section 2.4 the problem class $\mathcal{NP}$-hard is defined. It is known that the VRPTW is $\mathcal{NP}$-hard [5], and since VRP is a special case of VRPTW it is also $\mathcal{NP}$-hard. In addition the TSP is also $\mathcal{NP}$-hard.

It is not possible to state which of the models presented is the best to model the pack machine problem. In this section however, a qualitative comparison between the different models is made. From an academic point of view, an ideal formulation, is the one which has as small a solution space as possible, while no feasible solutions are eliminated [4], this is known as the *convex hull*. This is important when using exact methods, since these tend to terminate faster, when the solution space is precisely described. It is for example rather easy to find an optimal solution for an integer problem, if the $LP$-relaxation has an integral optimum. When dealing with a real-life problem as the pack machine problem or the limited pack machine problem, there could though be several aspects when evaluating the model.

The VRP model has a big solution space. It is rather easy to find an initial solution, and to improve this solution. However, because of the big solution space, it is rather difficult to find an optimal solution. The problem is divided into two smaller instances, which seems good from a computational point of view, since the solution space is reduced considerable. Better overall solution could though be obtained, when focusing on the cost, if the problem was regarded as one big problem instead of two smaller problems.

For the VRPTW model it is $\mathcal{NP}$-hard just to find a feasible solution if the number of vehicles is limited, because of the time windows according to [6]. The solution space is smaller than for the VRP model, as the structure of the time windows, removes some of the worse solutions. These time windows are made from the original solution provided by Danisco, and this solution is supposed to be good. Compared to the VRP model, this VRPTW model is superior according to handling the customers which should be serviced twice. The time windows guarantee, that these visits will appear as two distinct visits. The time windows will force the customers in the northern part of Sweden to be serviced first, and each double visit not to be serviced sequently. The time windows will however discard some feasible solutions, which is a problem, if the actual optimal solution is among those. In Section 2.5.1 the problem by modelling these double visits is described further.

The two models could be combined, such that only customers in the northern part of Sweden and customers which should be visited twice, has a time window. This is however not done in this thesis, but it could be a subject for further investigation.

The double TSP model seems like a good approach for solving the limited pack machine problem described in Section 1.6. The idea of dividing the problem in two, has the same advantages and disadvantages as the divided VRP model. Since there are only one pack machine available for each of the sets of customers, the solution space is much smaller than for the VRP model.

### 2.5.1 Formulation of Double Visits

Two different approaches have been proposed to solve the problem with the customers which should be visited twice, that is a formulation with time windows and a formulation, where they are visited in the same order in the two tours. Both formulations have the disadvantage, that the optimal solution to the original problem could be discarded from the solution space. The idea behind a time window constraint, is to separate the two visits from each other, but the time windows also discards solution, where there are the requested time period between the two visits, but for some reason the time windows are not adhered. In the other model, where customers are visited in the same order, better solutions might be obtained, if the customers were visited in another order, while the double visits were divided as they should. The key observation is here, that the double visits could be modelled, such that a given time period should be between the two visits. By using the following notation, this could be modelled:

Let $\mathcal{C}_d$ be a subset of $\mathcal{V}$ consisting of the customers which should be visited twice. The set $\mathcal{C}_d$ can be divided in the subsets $\mathcal{C}_{di}$ such that $\mathcal{C}_d = \mathcal{C}_{d1} \cup \mathcal{C}_{d2} \cup ... \cup \mathcal{C}_{dn_c}$ and $\mathcal{C}_{di} \cap \mathcal{C}_{dj} = \emptyset \forall i, j$, where $n_c$ is the number of subsets. Each of the subsets contains a customer which should be visited twice. Such a customer is modelled as two customers instead of just one. For the subset $\mathcal{C}_{di}$ the customers are noted as $i_1$ and $i_2$. The parameter $w_{i1}$ states the arrival time at customer $i_1$ and the service time at customer $i_1$ is noted as $s_{i1}$, analogue to the model presented in 2.3. Let $T_{i1i2}$ state the temporal distance between the two visits, that is:

$$T_{i1i2} = \begin{cases} M & \text{if } w_{i2} \geq w_{i1} \\ w_{i2} - (w_{i1} + s_{i1}) & \text{otherwise} \end{cases}$$

The big $M$ notation is used for a sufficiently high number, which is used when customer $i_2$ is visited before $i_1$. In Figure 2.1 the situation is illustrated, where $i_1$ and $i_2$ is two visits for the same customer and $T_{i1i2}$ is the time between the two visits.

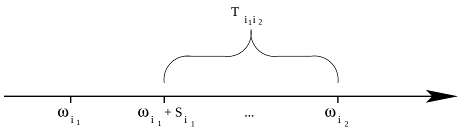



Figure 2.1: Timeline for double visits.

If $L$ defines the minimum allowed time in between such two visits, the constraint in (2.24) will guarantee that this is fulfilled.

$$T_{i1i2} \geq L \tag{2.24}$$

This approach is however not used in neither the solution methods nor in the implementation.

Chapter 3

# Literature Review

The two problems treated in this project are the pack machine problem and the limited pack machine problem, which, as far as I know, are not treated in the literature. The problems used in the approach to model the pack machine and the limited pack machine problems are on the other hand among the most well studied optimization problems. Hence focus will be on these problems in the literature review presented in this chapter. The amount of literature on VRP, VRPTW and TSP is enormous, and it is beyond the scope of this thesis, to make a complete review of the existing literature concerning these topics. The focus will be on the literature used in this project. A review of solutions methods developed for the problems is given in Section 3.1, 3.2 and 3.3.

## 3.1 VRP

In [3] a comprehensive review of literature on the VRP is given. Different mathematical models of the problem is given, and variants of the problem are also treated. Literature on three types of solution methods for the VRP, heuristics, metaheuristics and exact methods, are presented briefly in the next paragraphs.

#### 3.1.0.1 Heuristics

The heuristics are some of the first solution methods developed for the VRP, as some of these are developed in the sixties. Two of these methods are described in this thesis, that is the saving algorithm and the sweep algorithm. The saving algorithm was developed by Clarke and Wright and is described in many other articles, as e.g. in [3] and [7]. As it is designed for the capacitated vehicle problem, it does not seem straight forward to apply the method on the pack machine problem, as there are no limitations on capacities in the pack machine problem. The sweep algorithm on the other hand seems more applicable to the pack machine problem. It is not clearly defined, who developed this method as stated in [3]. In the first methods by for example Gillett and Miller the customers were clustered according to their geographical coordinates, whereas in later versions of the algorithm, the customers were clustered by solving a generalized assignment problem as described in [3]. The improvement methods are either single route improvements or multi route improvements, single route improvement methods focus on the single routes, where multi route improvement focus on the entire solution. Single route improvements are the same improvement methods as for the TSP, and these will be reviewed in Section 3.3. In [7] some multi route improvement methods are presented, that is the *string cross*, the *string exchange* and the *string relocation* methods. These are described further in Section 5.2.6.

#### 3.1.0.2 Metaheuristics

There are 6 types of metaheuristics which have been applied to the VRP, according to [8]. In this review only literature on Simulated Annealing and tabu search is reviewed. This is done to keep focus on the methods implemented or described later in this thesis.

In [9] a comparison between the performance of Steepest Descent and Simulated Annealing is presented. Three neighbourhoods have been used, that are *string relocation*, which relocates a string from one route to another, *string exchange*, where strings are exchanged between two routes and *string mix*, which combines the two other neighbourhoods. The most important results are, that Simulated Annealing is superior to Steepest Descent, and tabu search yields better results than Simulated Annealing. There are more literature concerning tabu search than on simulated annealing. The earlier work includes [10], who uses the $\lambda$-interchange generation as neighbourhood. Later works include the granular tabu search presented in [11]. The method uses the insight, that long edges are rare in optimal solutions, and hence solutions which contains these edges are eliminated, the results of this method are impressive as good solutions are found fast.

#### 3.1.0.3 Exact Methods

In [3] a review of the exact methods for the capacitated vehicle routing problem is given. The developed methods are based on branch and bound, where the bounds come from relaxations. The capacity cut constraints could be omitted in the relaxed model, other relaxations are based on the *K-tree* relaxation, the counterpart to the *1-tree*-relaxation for the TSP. Better bounds are obtained by lagrangean relaxation or by solving the dual of a set partitioning formulation of the problem. Branch and cut methods have also been developed for the VRP, see for example [3]. A lot of attention is devoted to branch and cut, as it has been very successful in finding optimal solutions for the TSP.

## 3.2 VRPTW

A comprehensive review of solution methods for the vehicle routing problem with time windows is given in [12] and [13]. In [12] construction heuristics and local search algorithms are presented, while literature on metaheuristics for the VRPTW is presented in [13].

#### 3.2.0.4 Heuristics

Early literature on construction heuristics includes [14], where a nearest neighbour construction heuristic based on a weighted cost function is presented, where the weighted cost function is calculated by including a measure of temporal distance between customers, another measure which is included estimates the urgency of a customer to be serviced according to a given time. In addition a time oriented saving heuristic and insertion heuristic is presented in this article, where the best results are obtained with the insertion heuristic. The routes are scheduled sequentially in the nearest neighbour and the insertion heuristic, which could be disadvantageous, as described in Section 10.3.2. In [15] a parallel variant of the insertion heuristic is presented, this improves the performance of the heuristic. The improvement methods are mainly based on the methods used with VRP, such as $\lambda$-exchange and OR-opt exchanges, these methods are described further in Section 5.2.6.

#### 3.2.0.5 Metaheuristics

There are a lot of literature concerning tabu search for the VRPTW, there are on the other hand not much literature on simulated annealing for the VRPTW. A brief review of metaheuristcs for the VRPTW is given in [3], while a more comprehensive is given in [13]. The improvement methods described in the previous paragraph can constitute the neighbourhood for the metaheuristics.

In [16] both a Simulated Annealing, a tabu search and a genetic algorithm is presented. The tabu search and the genetic algorithm yields the best solutions, but Simulated Annealing is faster according to the computation time. The $\lambda$-exchange operator makes it out for the neighbourhood in the implementation of the methods. The idea of combining a local search algorithm with Simulated Annealing is interesting, and could be applied on the pack machine problem.

In [17] a parallel version of Simulated Annealing is developed, where different processes of Simulated Annealing co-operates, such that better solutions according to the cost could be obtained, the *relocate operator* is used as neighbourhood, this neighbourhood is described further in Section 5.2.6.1. The results are impressive as the best known value of the cost function were found for most of the test instances.

#### 3.2.0.6 Exact Methods

There are several articles concerning exact methods for the VRPTW. Most of the methods are variants of the methods used for the VRP. Two articles [18] and [5] have been studied further in connection with this project.

In [18] a column generation method based on the linearly relaxed set partitioning model is developed for obtaining a dual bound to the VRPTW. Furthermore some branching strategies are presented for the branch & bound method.

In [5] the dual bounds obtained by the column generation are improved with a *2-path cut* procedure. Both articles are interesting, as they could be applied to the pack machine problem, though it might cause some difficulties with the double visits.

## 3.3 TSP

The TSP is indeed one of the most studied optimization problems. But as the TSP is a minor part of this project, only a very brief literature review will be carried out on this subject. In article [19] a comprehensive review is made on the TSP. This review includes descriptions of related problems and practical applications. The construction heuristics described include nearest neighbour and insertion heuristics, heuristics based on trees and saving methods. The improvement heuristics includes the 2- and 3-opt methods. Only a brief review of the metaheuristics such as Simulated Annealing and tabu search is given. Methods for finding lower bounds include the 1-tree relaxation and subtour relaxation. Different types of cuts, which could be used in a cutting plane algorithm, are presented, among those are the subtour elimination inequalities and the comb inequalities. Application of exact methods such as Branch & Bound and Branch & Cut to the TSP is also presented, where the best results are obtained by Branch & Cut.

Chapter 4

# Data

There is only one data set used in this project. It consists of informations about the Swedish customers, which should have bulk delivered. There are 570 customers in the data set. The data set is provided by Danisco, and it contains following informations about the customers. The data set is seen in Appendix A.2. 288 of the customers are located in Scania.

*Ship to* A unique parameter for each customer, represented as an integer.

*Name* The name of the customer. This is not used in the project.

*ID* Another unique parameter for each customer, represented as an integer.

*X-coord* The geographical position of the customer, *X-coord* is an integer representing the latitude, at which the customer is located.

*Y-coord* The geographical position of the customer, *Y-coord* represents the longitude.

*Machine nr.* is an integer taking the value one, two or three indicating which machine, the customer is serviced by.

*G.zo* is an integer representing the geographic zone, Scania is represented as 401, and 402 to 404 represents the rest of Sweden.

*Dlv. date* indicates the time windows for the solution. The campaign is divided in two periods each represented by a date. The date `01.09.2005` indicates, that the customers should be served in the first part of the campaign, that is until the 15. of November, while `05.09.2005` indicates that the customers should be serviced in the last part of the campaign. Some of the customers has another *dlv. date*, for example `28.09.2005`, these are the customers, which arrived after the start of the campaign. In these cases, the date represents the actual date of delivery. The number of customers, which has another *dlv. date* than `01.09.2005` or `05.09.2005` can be used to estimate how much of the problem is static and how much is dynamic. The number of customers which has another dlv. date is 46, hence 8% of the problem is dynamic.

*Order qty.* is the amount of bulk, demanded by the customers.

Danisco provided another data set containing the mentioned informations and in addition some handwritten dates. These handwritten dates are the actual delivery dates from the campaign in 2005. This data set is also ordered, according to the actual routes, and hence it provides the original solution used by Danisco in 2005. It should be noted, that the two data sets contain the same customers.

The position is given in longitude and latitude coordinates. In this project, it would be appropriate, if the geographic positions of the customers are represented by UTM-coordinates, since the distance between two points in UTM-coordinates corresponds to the distance in meters [20]. The procedure for converting the coordinates to UTM, is provided in [21].

In Figure 4.1, the distribution of the Swedish customers is seen, they are depicted by blue stars. The positions are represented by UTM-coordinates. The pattern of the customers clearly indicates the shape of Sweden. It is furthermore seen, that the customers are mainly placed in the southern and western part of Sweden, that is mainly in the Scania region. The island *Øland* stands out. Indicating that many customers are located at *Øland*, and by the coast near Kalmar.

In Figure 4.2 the original solution provided by Danisco is depicted. There are three routes, the first one handles the customer in the western part of Sweden, The second handles the customers in Scania and the middle of Sweden, while the last one handles the customers located in the eastern part of Sweden. The island *Øland*, stands even more distinct out, than at Figure 4.1. The island stands out, since it is observed, that the tractor visits the customers in *Øland* in two periods, and for each of these periods, the tractor does not return to the mainland. It is seen that the vehicle crosses the bridge from the mainland, and then visits all the customers at *Øland*, which are mainly located at the

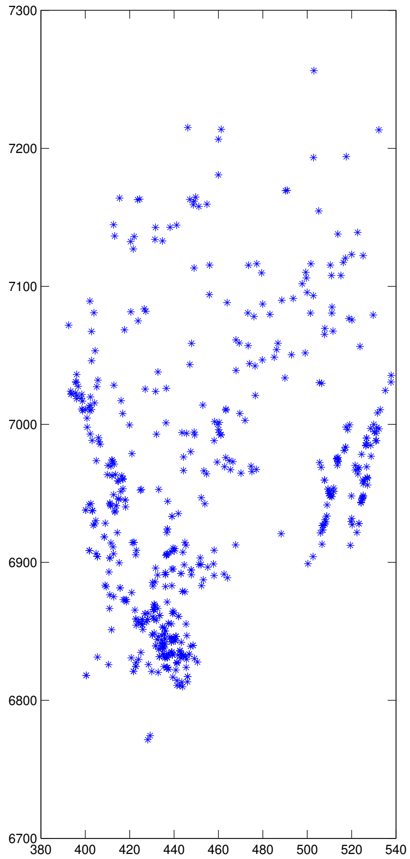



Figure 4.1: Distribution of the Swedish customers, the units of the horizontal and the vertical axis are respectively the eastern and northern coordinates represented as UTM coordinates divided by 1000.

coast, before returning to the mainland via the bridge again. When solving the problem, the distances are assumed to be euclidean. An example of a problem occurring when using euclidean distances is seen at *Øland*, since the formulation of the problem does not considerate, that vehicles has to use the bridge, and can not use the sea as a road. It seems like one part of the route does not cross the bridge, this is however not the case, as the line indicating the routes is the direct line between customers, and not the actual route driven by the pack machines.

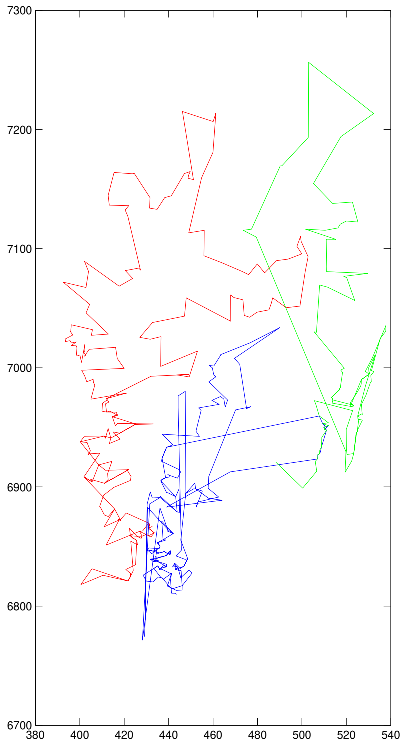



Figure 4.2: The original solution provided by Danisco, the units of the horizontal and the vertical axis are respectively the eastern and northern coordinates represented as UTM coordinates divided by 1000.

The time windows depicted in Figure 4.3 are calculated from the delivery date in 2005 according to the solution provided by Danisco. They are calculated as $+/-5$ days from the date they were serviced in 2005. There is a special structure of the time windows. This is because, the order of the customers is obtained from the routes in the solution provided by Danisco. It is seen, that the first 187 customers and the customers between 415 and 462, constitute one route. The second route consists of customers between number 188 and 317, and between number 465 and 548. While the rest of the customers belong to route 3. These time windows are used in the VRPTW approach.

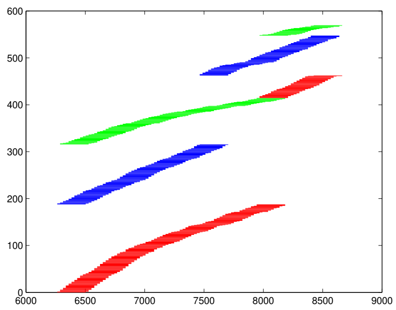



Figure 4.3: Time windows for the customers, the horizontal axis is the time, the vertical is the customers. Each colour corresponds to one route.

CHAPTER 5

# Solution Methods

In this chapter the solution methods used to solve the different models of the problem are described. Some of the solution methods described are not implemented, but are included, for the sake of completeness.

## 5.1 Solution Methods for the VRP Formulation

The VRP formulation is described in section 2.1. The customers are divided in two sets, the first containing customers in the northern part of Sweden, as these should be visited first, the other contains the customers in the central and southern part of Sweden.

In this section two heuristics to obtain an initial feasible solution to the approach based on the VRP formulation are described. The VRP formulation of the problem is the one described in section 2.1. The second heuristic is implemented, and the results will be presented and treated in section 8.1.

The first heuristic are based on Clarke and Wright's *saving heuristic* described in [7], while the second is developed from a two phase heuristic called the *sweep algorithm*, which is described in [3]. It is noted, that the formulation in Section 2.1 does not handle the double visits, and the solution methods described in this section will also not handle the double visits. The methods have been chosen, as they are simple but still provide solutions, which have an acceptable quality according to the cost of the obtained solutions.

### 5.1.1 A Saving Based Heuristic

The main idea behind this method is, that if two routes $(0, ..., i, 0)$ and $(0, j, ..., 0)$ in a closed VRP can be combined to one route, then the saved distance, called *saving* can be defined as:

$$save_{ij} = c_{i0} + c_{0j} - c_{ij} \tag{5.1}$$

#### 5.1.1.1 The Problem for the Customers in Northern Sweden

The VRP based model can be interpreted as follows. The depot is the depot for the pack machines, which is located in Scania, i.e. very far from these customers. The set of customers in northern Sweden is $\mathcal{V}_{north}$. The number of customers in northern Sweden is $n_{north}$, whereas $T_{north}$ denotes the time to be used in the northern part of Sweden. How $T_{north}$ can be determined is a subject of further investigation. It could be fixed empirical, by trying different values of $T_{north}$. The value should be low, but still large enough, such that a feasible solution can be obtained, by using the available pack machines. The set of routes are denoted $\mathcal{R}$, each of the routes corresponds to the vehicles available in $\mathcal{K}$.

The saving based heuristic is presented in the following, and a pseudo code for the heuristic can be seen in Algorithm 1.

In line 1 to 5 in Algorithm 1 the routes are initialised, there are $n_{north}$ different routes, each containing exactly one customer. The length of the routes, measured in time are calculated too. There is a problem, because you do not actually know the service time $s_{ik}$, since you do not know which vehicle services which customer. This could be solved by using an upper bound. This upper bound could be calculated by assuming that the pack machine with the lowest effectivity is servicing all customers.

The savings are calculated in line 6 to 8 and sorted in a non-decreasing list in line 9. A sorting algorithm like for example heap sort could be used to sort these, heap sort is described in [22].

In line 10 a while loop is started in order to merge routes as long as the part

**Algorithm 1** Pseudo code: Saving heuristic applied in northern Sweden

1: **for** $i = 1$ to $n_{north}$ **do**
2: $\quad route_i := (0, i, 0)$
3: $\quad \omega_i := t_{0i} + t_{i0} + s_{ik}$
4: $\quad \mathcal{R} := \mathcal{R} \cup route_i$
5: **end for**
6: **for** $(i, j) = 1, ..., n$ **do**
7: $\quad save_{ij} := c_{i0} + c_{0j} - c_{ij}$
8: **end for**
9: Sort savings in non-increasing list
10: **while** $\max \{\omega_i : i \in \mathcal{V}_{north}\} \leq T_{north}$ **do**
11: choose next saving $save_{ij}$ in the list
12: **if** there exist two routes, $route_u$ containing edge $(0, j)$ and $route_v$ containing edge $(i, 0)$, that can be feasible combined **then**
13: $\quad\quad route_{new} := (..., i, j, ...)$
14: $\quad\quad \omega_{new} := \omega_u + \omega_v - t_{0i} - t_{j0} + t_{ij}$
15: $\quad\quad \mathcal{R} := \mathcal{R} \cup route_{new}$
16: $\quad\quad \mathcal{R} := \mathcal{R} \setminus \{route_u, route_v\}$
17: **end if**
18: **end while**
19: return $\mathcal{R}$

of the campaign length for the northern part of Sweden is not exceeded. The routes are merged in line 11 to 14, this is done by removing two old routes from $\mathcal{R}$, which can feasibly be merged into one new route, and then adding the new route to $\mathcal{R}$. The time duration of the routes are updated too.

The complexity of the algorithm can be calculated in the following way. The operations in line 2 to 4, which can be performed in constant time are carried out $n_{north}$ times. There are $n_{north}(n_{north} - 1)/2$ different node pairs, in the symmetric version of VRP, for which a saving should be calculated. If the distances between customer $i$ and $j$ are the same as between customer $j$ and $i$, the VRP is said to be symmetric. If the VRP is symmetric, as is the case in this project, the savings $save_{ij}$ will equal $save_{ji}$, and only half of the savings must be calculated compared to the non-symmetric case. The sorting in line 9 can by using a sorting algorithm such as heap sort be performed in $\mathcal{O}(n^2_{north}/2 \log(n^2_{north}/2))$ flops. Initially there are $n_{north}$ routes, and in the best case all these routes are merged to one big route. That is, there are at maximum $n_{north} - 1$ merging operations. A merging operation can be carried out in constant time, hence the while loop in line 10 to 18 adds $n_{north} - 1$ flops to the complexity. Adding up the complexity gives following expression: $n_{north} + n_{north}(n_{north} - 1)/2 + n^2_{north}/2 \log(n^2_{north}/2) + n_{north} - 1$. The dominating term in this expression is $n^2_{north}/2 \log(n^2_{north}/2)$, which yields a total complexity at $\mathcal{O}(n_{north}^2 \log(n_{north}^2))$ for the Saving Heuristic.

#### 5.1.1.2 The Problem for the Customers in the Central and Southern Part of Sweden

The number of remaining customers, that are the customers located in the central and the southern part of Sweden is denoted as $n_{rest}$, the set of these customers is denoted $V_{rest}$, whereas the part of the campaign length, which should be used in this area are called $T_{rest}$. There are two different depots in this part of Sweden, a start depot $d_k^+$ and an end depot $d_k^-$. Algorithm 1 can be used to solve this problem if the routes are initialised as $(d_k^+, i, d_k^-)$ for $i = 1, ..., n_{rest}$.

It is not guaranteed, that the solution found by the heuristic is feasible, as there is a limit of the number of pack machines, which can be used in the solution. This is for example the case, if more than 3 vehicles are used in the solution. One possibility will be to increase the time used in either the northern or the central and southern part of Sweden. The obtained solution is probably not optimal, and one can choose to improve the routes by some of the improvements methods described in Section 5.2.6.

This heuristic is not implemented for neither the customers in the northern part of Sweden nor for the customers in the central and the southern part of Sweden. The heuristic is not considered anymore.

### 5.1.2 A Heuristic Based on the Two-Phase Method

This algorithm is based on the sweep algorithm, which is a well known heuristic described in [7]. It is a cluster first - route second method. I.e. the customers are split into different groups, before the routes are scheduled. This operation results in several TSPs which should be solved, either to optimality or by another heuristic.

#### 5.1.2.1 The Problem in Northern Sweden

When using this method, we could assume, that there are $m = 3$ different depots in the southern part of the northern part of Sweden, the three depots correspond to the three available pack machines. This makes sense, since we would group the customers in three groups according to their geographical location. The depot for vehicle $k$ is denoted $d_k$, where $d_k^- = d_k^+$. A customer $i$ is represented by the coordinates $e_i, n_i$. The customers with $e_i \leq pos_{west}$ are classified as the *western* customers, customers with $e_i \geq pos_{east}$ are classified as the *eastern* customers and the remaining customers are classified as the *central* customers. The group of customers that belongs to a vehicle $k$ are called $\mathcal{G}_k$, the given geografical interval for this group are called $pos_k$. The effectivity of the pack machines should be taken into account, when dividing the customers, such that the pack machine with lowest effectivity services fewer customers than the other pack machines. The algorithm is formulated as pseudo code in Algorithm 2.

**Algorithm 2** Pseudo code: Two-phase heuristic applied in northern Sweden

1: **for** $k = 1, ..., m$ **do**
2: $\quad\mathcal{G}_k := \emptyset$
3: $\quad$**for** $i = 1, ..., n_{north}$ **do**
4: $\quad\quad$**if** $e_i \in pos_k$ **then**
5: $\quad\quad\quad\mathcal{G}_k := \mathcal{G}_k \cup i$
6: $\quad\quad$**end if**
7: $\quad$**end for**
8: **end for**
9: **for** $k = 1, ..., m$ **do**
10: $\quad$return solution for TSP to belonging to vehicle $k$
11: **end for**

The customers are grouped in line 1 to 8 in Algorithm 2 and the resulting TSPs are solved in line 9 to 12. How the TSPs are solved will be discussed in Section 5.1.3. The procedure for dividing the customers has a complexity of $\mathcal{O}(mn_{north})$. The complexity of the procedure for solving the TSPs are difficult to calculate. It depends on whether the TSPs are solved to optimality (in which case there are no known polynomial algorithms), or just solved by a heuristic.

#### 5.1.2.2 The Problem in the Central and Southern part of Sweden

In this part of Sweden it might make more sense, if the customers are grouped in another way, and not just according to their eastern coordinates, since the end depot is geographically located more central. The heuristic is only implemented for customers in the northern part of Sweeden, and for customers which should be serviced twice to illustrate the inexpedient manner, these are treated with in this approach.

### 5.1.3 Solving TSP with a Heuristic

An essential part of the algorithm described in Section 5.1.2 is to solve a Travelling Salesman Problem. The TSP is described and modelled in Section 2.3, where it constitutes a basis for the limited pack machine problem. In this section a construction heuristic and an improvement procedure for the Travelling Salesman Problem is presented. The construction heuristic *nearest neighbour* is used to obtain a feasible solution for the travelling salesman problem. The feasible solution is improved by using the *2-opt procedure*.

#### 5.1.3.1 Nearest Neighbour Heuristic

The used heuristic *nearest neighbour* is quite simple both to understand and to implement, the complexity is low, but it has a great disadvantage since the obtained solutions can be rather bad. The idea is to construct a tour by continuously adding the customer to the tour which is closest to the endpoint of the existing tour. The pseudo code for this heuristic can be seen in Algorithm 3, recall that the set of vertices is called $\mathcal{V}$.

**Algorithm 3** Pseudo code: The nearest neighbour construction heuristic

$\mathcal{W} := \mathcal{V}$
$j :=$ arbitrary vertex in $\mathcal{W}$
$l := j$
$\mathcal{W} := \mathcal{W} \setminus \{j\}$
**for** $\mathcal{W} \neq \emptyset$ **do**
    Find $j \in \mathcal{W}$ such that $c_{lj} = \min\{c_{li} | i \in \mathcal{W}\}$
    connect $l$ to $j$
    $\mathcal{W} := \mathcal{W} \setminus \{j\}$
    $l := j$
**end for**

This algorithm proceeds good in the beginning, where short arcs are added to the solution. Unfortunately it might miss some vertices which are located nearby the chosen vertices, since these are not located nearest to the end vertex. These will later be added at a high cost. The situation is illustrated in Figure 5.1, where the customer $k$ is missed as it is not the nearest neighbour, it is added later at a high cost.

Therefore some rather poor solutions with respect to the objective value can be found using this algorithm. The errors of this type might be great, but there are few of them, as stated in [19], therefore this algorithm might perform satisfying when combined with an improvement procedure.

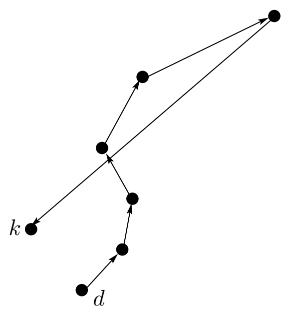



Figure 5.1: Illustration of the nearest neighbour construction heuristic.

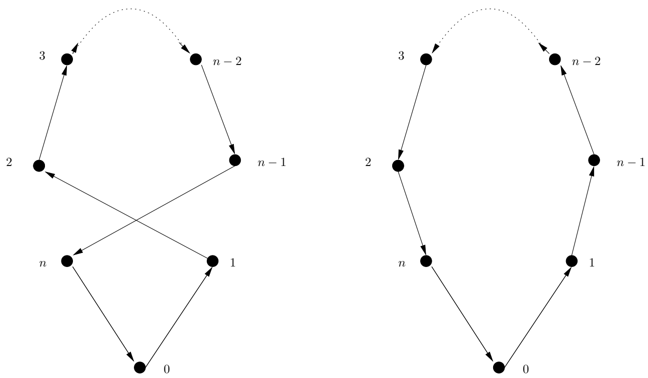
The complexity can be described in the following way: The for loop is performed $n-1$ times, where $n$ is the number of customers in the TSP. The number of customers, which are not included in the partial TSP solution is $n-1$ in the first iteration, $n-2$ in the next, and so on. This adds up to the sum $(n-1)n/2$, and hence the complexity is $\mathcal{O}(n^2)$

The algorithm can be applied to the problem in northern Sweden without modifications, as it is not crucial which vertex is chosen as the initial vertex, even though the performance of the heuristic depends on which node, that is chosen initially.

For the customers in the central and southern part of Sweden it would not be reasonable to choose the initial vertex random, instead either the depot or the customer located northernmost should be chosen.

#### 5.1.3.2 Improving the Solution with the 2-opt Procedure

The TSP-solutions can probably be improved by changing parts of the route. The method used to change parts of the route is called the *2-exchange*-operation. Figure 5.1.3.2 illustrates how an 2-exchange operation can be performed. When looking at the routes, a 2-exchange operation corresponds to exchange two edges, which does not have any incident vertex in common. In Figure 5.1.3.2 the 2-exchange operation has been performed with the arc between vertex 1 and 2 and the arc between vertex $n-1$ and $n$. It is noted that the direction of the route is changed for the customers in between customer 2 and $n-1$.




(a) Route before 2-exchange operation

(b) Route after 2-exchange operation

Figure 5.2: Example of a 2-exchange operation, the route has the following sequence of nodes: $[0, 1, 2, 3, ..., n-2, n-1, n, 0]$. After the 2-exchange operation, the route has following sequence of nodes: $[0, 1, n-1, n-2, ..., 3, 2, n, 0]$.

In general a 2-exchange operation between two arbitrary arcs between vertex $i$ and $j$ and vertex $k$ and $l$, $(i \neq j \neq k \neq l)$ can be described as following change in the route sequence:

Route sequence before 2-exchange operation
$[0, 1, ..., i, j, j+1, ..., k-1, k, l, ..., n]$

Route sequence after 2-exchange operation
$[0, 1, ..., i, k, k-1, ..., j+1, j, l, ..., n]$

When the route can not be improved anymore by use of the 2-exchange operation, it is called *2-optimal*. It can be investigated whether the solution is 2-optimal in $\mathcal{O}(n^2)$ as stated in [19].

## 5.2 Solution Methods for VRPTW-formulation

The problems are solved with two solution methods. A local search heuristic named *Steepest Descent* and a metaheuristic named *Simulated Annealing*.

Metaheuristics and local search methods are general solution methods, which can be used in various optimization problems. They are frameworks, i.e. there are given some guidelines, to how the solution could be improved. The metaheuristic or the local search method has to be adapted to the specific problem.

Neither of the methods guarantee to give an optimal solution. But experiments have shown, that a metaheuristic often gives a good solution, and the solution is better than local search methods, even though the solution is not necessary optimal, it is often only 5% worse than the optimal solution according to [8]. A metaheuristic is often used, when the problem is $\mathcal{NP}$-hard, because there are no known method yet, which can solve the problem in polynomial time. Therefore it is often preferable, to obtain an near optimal solution in much less time, than obtaining an optimal solution with an exact method which might be slower. Another advantage of both local search and metaheuristic is that common programming languages like Java or C++ can be used to implement them. This is especially important, when the method should be used by a concern, or whenever economic issues are of matter, since the license to many optimization programs are really expensive, compared to software only based on Java or C++.

The metaheuristic chosen to solve this problem is Simulated annealing and the local search method is Steepest Descent. In Section 5.2.1 the main concepts behind Simulated Annealing are discussed, and the general algorithm is presented. Steepest Descent is described in Section 5.2.2. Sections 5.2.3, 5.2.4, 5.2.6, 5.2.7 and 5.2.8 describe how the metaheuristic and the local search method are applied to the special problem.

### 5.2.1 Simulated Annealing - Main Concepts

Simulated Annealing is named after a concept in statistical physics. The idea behind the Simulated Annealing is invented from the observed results when iron sheets are cooled. Simulated Annealing is described in various articles such as [23].

When using Simulated Annealing, one have to start with an initial solution $S$ to the problem. The initial solution can be found by a construction heuristic, how this is done, is described further in Section 5.2.3. Another solution $S'$ is found randomly in the *neighbourhood* to the initial solution. The solution $S'$ is a candidate to become the new chosen solution $S$. The neighbourhood consists of solutions, which are defined from the current solution (the initial solution). A discussion of how the neighbourhood can be chosen to this problem is presented in Section 5.2.6. The solution $S'$ is evaluated, and if it is better than the old solution $S$, it is chosen as the new solution $S$. If the solution $S'$ is worse than $S$, it is accepted as a new solution with the probability $e^{-\Delta/T}$. Where

$\Delta$ defines how much better $S$ is than $S'$, that is $f(S) - f(S)'$. Where the function $f$ defines the quality of a given solution. How this function is defined is described in Section 5.2.7. The parameter $T$ is called the temperature. The initial temperature is chosen before the start of the algorithm. The temperature decreases as a function of how many iterations the metaheuristic has performed. Therefore the probability of choosing a worse solution as a new solution decreases as a function of how long time the metaheuristic has been performing. The probability of choosing a worse solution as new solution depends also on how much worse the solution $S'$ is. The metaheuristic terminates, when a *stopping criterion* is reached. A discussion of which stopping criterion to use is found in Section 5.2.8.

Choosing solutions which are worse than the best found solution is a way of avoiding local optima. A local optimum is a solution, where no better solutions are available in the neighbourhood. It is though not given, that the local optimum is the best solution to the problem, which is called the global optimum. The attempt to avoid a local optimum is the difference between metaheuristics as Simulated Annealing, tabu search etc. and a simple local search method as Steepest Descent.

The pseudo code for Simulated Annealing is presented in Algorithm 4.

**Algorithm 4** Pseudo code: The metaheuristic Simulated Annealing

```
1: S := S_start
2: T := an initial temperature
3: r := the reduction factor (0 < r < 1)
4: L := number of iterations between each reduction of T
5: while no stopping criterion is reached do
6:    for i:= 1 to L do
7:       N(S) := neighborhood to S
8:       S := solution in N(S)
9:       Δ := f(S') − f(S)
10:      if Δ ≤ 0 then
11:         S := S'
12:      else
13:         S := S' with probability e^(−Δ/T)
14:      end if
15:   end for
16:   T := rT
17: end while
18: return S
```

Initially the parameters $S$, $T$, $r$ and $L$ are defined, line 1 to 4 in the algorithm.

A candidate solution $S'$ is found in the neighbourhood to $S$ (line 8), compared to the old solution (line 9), and accepted or rejected as new solution $S$ in line 10 to 14. This procedure continues until a stopping criterion is fulfilled, in which case the solution $S$ is returned.

The parameters $S$, $T$, $r$ and $L$ should obviously be defined such that the best solutions for the problem is obtained. One should use *parameter tuning* to obtain the best suited parameters for the problem. The idea is to try various combinations of parameters at some test data sets. The concept of parameter tuning and how it is performed in this project is discussed in more details in Chapter 7.

It does not make sense to investigate the complexity of Simulated Annealing, since it depends on the stopping criteria, the initial solution and the neighbourhood. Instead a complexity analysis will be carried out, when these concepts are described.

### 5.2.2 Steepest Descent

Steepest Descent is a solution method which does not have the ability to avoid a local optimum. The method is contained in this project, as it can identify, whether there are better solutions in the neighbourhood.

There are many similarities between Simulated Annealing and Steepest Descent, such that many of the features which is needed for Simulated Annealing, is also needed in Steepest Descent. The neighbourhood, the cost function and the initial solution are defined in the same way as those for Simulated Annealing. Steepest Descent is described further in [23].

From a given solution $S$, the whole neighbourhood is investigated. The best solution $S^*$ in the neighbourhood, according to the cost function, is chosen, if it is better than the cost of $S$. This procedure continues until, the best solution in the neighbourhood is worse than the present solution $S$. The method is called Steepest Descent since, it chooses the solution in the neighbourhood, which gives the greatest improvement, i.e. has the lowest cost value, when minimization is the objective. The pseudo code for the Steepest Descent method is given in Algorithm 5.

Steepest Descent terminates, when a local minimum is reached. The problem is, that the local minimum, is not neccesary identical with the global minimum, and can in fact be very far from the global minimum, according to the cost function. One of the advantages by using the Steepest Descent method is, that it identifies

**Algorithm 5** Pseudo code: The solution method: Steepest Descent

```
S := S_start
f(S) := cost of S
stop := false
while stop = false do
   N(S) := neighborhood to S
   S* := best solution in N(S)
   if f(S*) ≤ f(S) then
      S := S*
   else
      stop := true
   end if
end while
return S
```

if there are any better solutions in the neighbourhood. The method can therefore be used to investigate the neighbourhood, if Simulated Annealing tends to find only worse solution for a while. In theory, Simulated Annealing, should be able to obtain better solution than Steepest Descent, therefore the solutions obtained by Steepest Descent can be used as a reference for those obtained from Simulated Annealing. Another though not entirely positive difference between Simulated Annealing and Steepest Descent, is that when using the same initial solution and neighbourhood, the solution found by Steepest Descent will always be the same, where as the randomness of Simulated Annealing gives rise to more different solutions.

There are the same problems concerning the complexity analysis as with Simulated Annealing.

### 5.2.3 Obtaining an Initial Solution

An initial solution to VRPTW is obtained by using an *Nearest Neighbour* construction heuristic. It is quite similar to the heuristic described in Section 5.1.3.1. But this time it deals with the overall VRPTW and not only the TSPs. The time windows are also considered. The heuristic is described in [14].

The routes are extended continuously, one at a time, until all customers are scheduled in a route. The customer inserted to a given route is the customer located nearest to the end of the route. Nearest is not only measured in geographical distance, but also in a temporal distance, how this distance is measured can be seen in expression (5.5). When there are no more customers which can be scheduled feasible in the route, a new route is started, until all customers are scheduled. The notation for the heuristic is as follows:

$i$ is the last customer in the present partial constructed route.

$j$ is a customer which can be serviced directly after customer $i$.

$d_{ij}$ is the geographical distance between customer $i$ and $j$.

$e_{jk}$ states the earliest possible time vehicle $k$ can service customer $j$. This time is either the start of the time window for customer $j$ or the time customer $i$ is finished serviced plus the travelling time from customer $i$ to $j$, depending on which of them is highest. $e_{jk}$ is calculated in (5.2).

$$e_{jk} = \max\{a_j, e_{ik} + s_{ik} + t_{ij}\} \tag{5.2}$$

$T_{ijk}$ is the temporal distance, that is the time difference between customer $i$ is serviced to customer $j$ can be serviced, if vehicle $k$ service them both, it can be calculated using the expression in (5.3).

$$T_{ijk} = e_{jk} - (e_{ik} + s_{ik}) \tag{5.3}$$

$v_{ijk}$ is a measure of how urgent customer $j$ should be serviced after customer $i$, if vehicle $k$ service them both. The urgency of a given customer $j$ to be serviced depends on the time windows. It can be calculated using the expression in (5.4).

$$v_{ijk} = b_j - (a_i + s_{ik} + t_{ij}) \tag{5.4}$$

The travel times $t_{ij}$, time windows $[a_i, b_i]$, service time $s_{ik}$ are defined as in Chapter 2. The cost indicator $\hat{c}_{ijk}$ is calculated using the expression in (5.5).

$$\hat{c}_{ijk} = \delta_1 d_{ij} + \delta_2 T_{ijk} + \delta_3 v_{ijk} \tag{5.5}$$

The geographical distance $d_{ij}$, the temporal distance $T_{ijk}$ and the urgency $v_{ijk}$ all contributes to the measure of the distance, how these should be weighted is controlled by the parameters $\delta_1$, $\delta_2$ and $\delta_3$. The parameters $\delta_1$, $\delta_2$ and $\delta_3$ should fulfill the criteria given in (5.6) and (5.7).

$$\delta_1 + \delta_2 + \delta_3 = 1 \tag{5.6}$$

$$\delta_1 \geq 0, \delta_2 \geq 0, \delta_3 \geq 0 \tag{5.7}$$

The parameters $\delta_1$, $\delta_2$ and $\delta_3$ should be chosen such that the best solution is obtained. In this project parameter tuning is used to fix the parameters. The procedure is described and the results commented in Section 7.2.1.

**Algorithm 6** Pseudo code: The nearest neighbour construction heuristic, applied on the VRPTW

1: $\mathcal{W} := \mathcal{V}$
2: $\mathcal{C} := \mathcal{K}$
3: **while** $\mathcal{W} \neq \emptyset$ and $\mathcal{C} \neq \emptyset$ **do**
4: $\quad j :=$ initial node in $\mathcal{W}$
5: $\quad l := j$
6: $\quad k :=$ a vehicle $\in \mathcal{C}$
7: $\quad \mathcal{W} := \mathcal{W} \setminus \{j\}$
8: $\quad \mathcal{C} := \mathcal{C} \setminus \{k\}$
9: $\quad$ **while** customers can be added to the route $k$ **do**
10: $\qquad$ Find feasible $j \in \mathcal{W}$ such that $c^{ind}_{ljk} = \min\{c_{lik} | i \in \mathcal{W}\}$
11: $\qquad$ connect $l$ to $j$
12: $\qquad \mathcal{W} := \mathcal{W} \setminus \{j\}$
13: $\qquad l := j$
14: $\quad$ **end while**
15: $\quad$ connect $l$ to the initially selected node.
16: **end while**

The pseudo code for the algorithm can be seen in Algorithm 6.

The initial vertex is the depot. It is not important how far the machines are from the depot after serving the last customer, hence it does not matter which customer is serviced last. Unlike the VRP formulation of this problem, the resulting solution is to the overall pack machine problem, and not just the partial problems in northern part and in the central and southern part of Sweden.

There will occur a problem, if not all customers are placed in a route, and no more vehicles are available. In this case the while-loop in line 3 terminates, when not all customers are scheduled in a route, the obtained solution will in this case be infeasible.

The complexity of the procedure can be interpreted in the following way. Without regarding the calculations used as a measure of the distance, the complexity is the same as for Algorithm 3, that is $\mathcal{O}(n^2)$. The parameter $e_{jk}$, is calculated recursively, which seems to increase the complexity, this is not the case as the information for the last customer in the route is used.

The routes are constructed sequentially. This is however not implicit an appropriate way to do it. One consequence of a sequential construction heuristic, is that the first routes will be pretty good, the next slightly worse, and the last one might be bad, since only the difficult customers are left. One could say, that the first routes are implemented in greedy way, such that the last will end up as worse solutions. One way to conquer this behaviour, is to insert customers in a parallel manner instead of the sequential. This is done, by initialising all routes at start, and then take advantage of the possibility of adding a new customer to all routes, such that the customer inserted at each iteration, is the customer which is nearest to the end of one of the routes.

### 5.2.4 Improving the Initial Solution

When the nearest neighbour heuristic terminates, not all customers are necessarily scheduled. Hence the initial solution found, is not necessarily feasible, when only $m = 3$ pack machines are available. As described in Section 5.1.3.1 nearest neighbour heuristic tends to miss customers which are located close to the route, but is not exactly the nearest neighbour, especially at the end of the performance of the heuristic, as stated in [19]. Another heuristic would maybe be able to schedule some of these not scheduled customers. The heuristic used is the *insertion heuristic*, proposed by Solomon, [14]. The main idea is to insert not scheduled customers between two customers on a partial route. The arrival times need to be updated for the customers scheduled later, according to the position in the route and the start time for the customer, in the route assuming that the pack machines leave the depot as soon as possible and the distances between the customer are euclidean, such that the *triangle inequality* is fulfilled. Since the arrival times are delayed, when inserting the new customer in the route, a new feasibility check is needed. The time update and feasibility check is described in Section 5.2.5.

### 5.2.5 Updating Time and Checking for Feasibility

Solomon proposes the following method for updating the time in [14]. The method is called *Push Forward*. When a customer $j$ is inserted between customer $i-1$ and $i$ at route $k$, then the difference between the new and the old arrival time for customer $j$ is given by $w_{i,k}^{new} - w_{i,k}$, where $w_{i,k}^{new}$ is the new arrival time at customer $i$ after inserting customer $j$ immediately before customer $i$ at route $k$ and $w_{i,k}$ is the old arrival time at customer $i$. This time difference indicates how much the arrival time is delayed when inserting the new customer. Therefore the Push Forward operation for customer $i$ is defined as in (5.8). An illustration of the Push Forward operation is seen in Figure 5.3 Where customer $j$ is insertied between customer $i-1$ and $i$, the time should be updated for customer $i$ and the customers scheduled after that in the route.

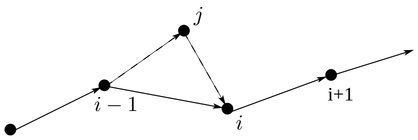



Figure 5.3: Illustration of the Push Forward.

$$PF_i = w_{i,k}^{new} - w_{i,k} \tag{5.8}$$

If there are euclidean distances between the customers, it is noted, that $PF_i \geq 0$. Recall, that if the pack machine arrives at a customer before the time window, it has to wait until the time window has started. The waiting time at customer $l$ is the difference between the earliest arrival time according to the time window $a_l$ and the actual arrival time $w_{l,k}$ when $a_l \geq w_{l,k}$. The waiting time at customer $l$, $wait_{l,k}$ can therefore be defined as in (5.9).

$$wait_{lk} = \max\{0, a_l - w_{l,k}\} \tag{5.9}$$

The delay at a customer $r+1$ is the same as the delay at customer $r$ subtracted the waiting time at customer $r$, if $r+1$ is the succeeding customer to $r$ on the route. The delay is always greater than zero and therefore the delay would be zero if $wait_{r+1,k} > PF_r$. This leads to the recursive formula in (5.10) for calculating the push forward for customer $r+1$, where $l \leq r < n_k$.

$$PF_{r+1} = \max\{0, PF_r - wait_{r+1,k}\} \tag{5.10}$$

It is seen, that if $PF_{r+1} = 0$ the delay for customers scheduled later than customer $r+1$ would also be zero, i.e. $PF_{r+2} = PF_{r+3} = ... = PF_m = 0$. It is also noted, that $PF_r$ is non increasing for the customers in the route.

The arrival time at customer $r$ can be calculated by using $PF_r$, if $l < r \leq n_k$ as seen in (5.11).

$$w_{r,k}^{new} = w_{r,k} + PF_r \tag{5.11}$$

The departure time can be updated in a similar manner, or by adding the arrival time and service time.

It is obvious, that if $PF_l > 0$, then it is possible, that the scheduling of some customers scheduled later than $l$ in the given route could be infeasible according to time. Feasibility can be checked by using the Push Forward operator. The scheduled customers in the route, where $l$ is situated, should be investigated sequentially until a customer $r$ is found where either:

1. $PF_r = 0$, the new route is feasible, since $PF_s = 0$ for $r < s \leq n_k$, and the routes were initially time feasible.

2. $PF_r > b_r - w_r$, the push forward of customer $r$ is infeasible, since the new arrival time is later than the end of the time window. The route can therefore not be a part for a feasible solution.

If the route is investigated, and there are no customers found, where either 1. or 2. is fulfilled, the route is feasible. This leads to the following necessary and sufficient condition for time feasibility, Lemma 5.1.

**Lemma 5.1** *Necessary and sufficient conditions for time feasibility, when inserting customer $l$ on the partially constructed route $k$, which is feasible:*

$$w_{l,k} \leq b_l \quad \text{and} \quad w_{r,k} + PF_{r,k} \leq b_r, \quad l < r \leq n_k$$

Customer $n_k$ is the last customer in the partial route, note that this is not necessarily the same as the depot. The condition $w_{l,k} \leq b_l$ guarantees that the inserted customer is time feasible, while the other condition guarantees that the customers scheduled later in the route are time feasible, after the insertion of customer $l$.

The push forward calculation has a complexity at $\mathcal{O}(n)$, where $n$ is the number of customers in the route $k$ which should be checked, and in the worst case scenario $n = n_k$. For each customer, the calculation can be taken out in constant time. The time feasibility check, has the same complexity as Push Forward, that is $\mathcal{O}(n)$.

### 5.2.6 Neighbourhood

The neighbourhood defines the way a given solution can be changed to another solution in the metaheuristic. There are several different neighbourhoods for the Vehicle Routing Problem, which can be applied to this particular problem. Some of the existing Neighbourhoods are described in the next paragraphs. The neighbourhoods are not implemented, the implemented neighbourhood is described at page 55.

### Examples of neighbourhoods to the VRPTW

The problem types VRP and VRPTW are quite similar, and therefore neighbourhoods defined to the VRP can also be used when solving VRPTW by a metaheuristic. A check for time feasibility is needed when adapting the neighbourhood to VRPTW. There are two main types of neighbourhood to the VRP. The first type can be categorised as the *Single Route improvement* methods. As the name says, these improvements methods only focus on the improvements of the single routes. Therefore these improvement methods can also be used to solve the TSP with a metaheuristic, the method will typically try to change the order of the customers in a given route. The second type can be categorized as the *Multi Route improvement* methods. Here the focus is not only on the single routes but on the complete solution. The methods will typically try to move customers from one route to another.

### Examples of single route improvement methods

The single route improvement methods $\lambda$-opt and Or-opt are described in various articles, see for example [7]. They will be described in the next.

**The $\lambda$ exchange operator** is the general version of the 2-exchange operator, which is described in Section 5.1.3.2. When using the 2-exchange operator, 2 arcs are removed from the route, and the 2 remaining segments are reconnected, without using the same arcs. A segment can be defined as in Definition 1.

**Definition 1** *A segment is a part of a graph, which is connected.*

When using the $\lambda$ exchange operator, $\lambda$ arcs are removed from a route, and the $\lambda$ remaining segments are reconnected again, in all possible ways, without using the same arcs. The best reconnection, that is the one yielding the shortest route, is chosen. A solution which can not be improved further by using the $\lambda$ exchange operator is called $\lambda$-optimal.

**The Or-opt-operator** is a method, where segments consisting of 1 or more consecutive nodes are removed from one part of the route, and reinserted between 2 other nodes. Figure 5.4 illustrates an example of an Or-opt-operation.

The next improvement methods are the *Swap-operator*, the *Re-insert operator* and the *Re-link operator*, these are actually designed as multi route improve-

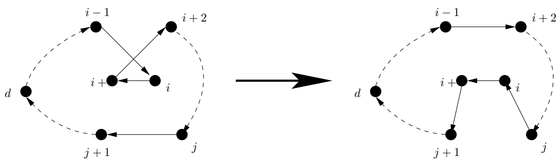



Figure 5.4: The route segment $(i, i+1)$ is removed from in between $i-2$ and $i+2$ and inserted between customer $j$ and $j+1$.

ments method in [24], but as they are described here, they are actual single route improvement methods. The improvement methods are described in [24].

**The Swap operator.** The 2 nodes $i$ and $j$ are removed from the route, and reinserted such that $j$ is inserted where $i$ was before and vice versa. An example of this procedure is seen in Figure 5.5

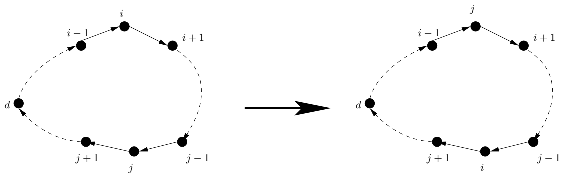



Figure 5.5: Example of the Swap operation, customer $i$ and $j$ change place. The route is before the swap-operation $(d, ..., i-1, i, i+1, ..., j-1, j, j+1, ..., d)$ and after $(d, ..., i-1, j, i+1, ..., j-1, i, j+1, ..., d)$

**The Re-insert operator.** A node $j$ is removed from the route and reinserted between two adjacent nodes $i$ and $i+1$. An illustration of this improvement method is seen on Figure 5.6

**The Re-link operator** is identical with the 2 exchange operator described in Section 5.1.3.2. It is worth noting, that the 2 exchange method is a single route improvement method, while the Re-link operator in [24] is designed to be a multi route improvement method.

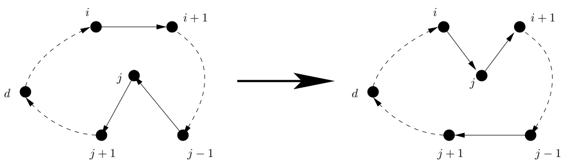



Figure 5.6: Example of the Re-Insert operation, customer $j$ is removed from the route, and reinserted between customer $i$ and $i+1$. The route is before $(d, .., i, i+1, ..., j-1, j, j+1, ..., d)$ and after $(d, .., , i, j, i+1, ..., j-1, j+1, ..., d)$

#### Examples of Multi Route Improvement Methods

In the following paragraphs, some multi route improvement methods are presented. The following improvement methods are described in [7].

**The Arc Cross operator.** An arc is removed from two of the routes. The partial routes are reconnected such that, one of the routes consists of the beginning of the first route combined with the end of the second route, and the other route consists of the beginning of the second route and the end of the first route. An example of the arc cross operation is illustrated in Figure 5.7.

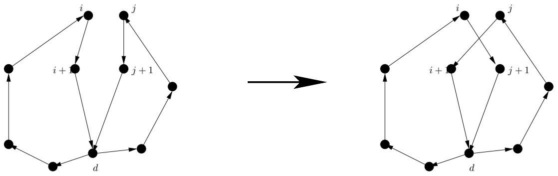


Figure 5.7: Example of the arc cross operation, the arcs $i, i+1$ and $j, j+1$ is crossed, such that the routes $(d, ..., i, i+1, ..., d)$ and $(d, ..., j, j+1, ..., d)$ are changed to the routes $(d, ..., i, j+1, ..., d)$ and $(d, ..., j, i+1, ..., d)$.

The arc cross operator is identical with the Re-link operator, if it is used as a multi route improvement method.

**The Arc Exchange operator.** In a feasible solution a route segment is removed from two of the routes. The route segments consist of maximum $k$ vertices. The route segment from the first route is inserted in the second route, and the route segment from the second route is inserted in the first route. An example of the arc exchange operator is seen in Figure 5.8.

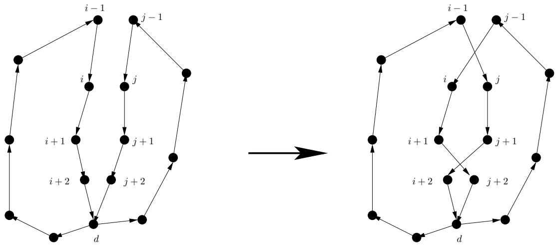



Figure 5.8: Example of the arc exchange operation. The route segments $(i, i+1)$ and $j, j+1$ is exchanged. Such that the routes $(d, ..., i-1, i, i+1, i+2, ..., d)$ and $(d, ..., j-1, j, j+1, j+2, ..., d)$ are changed to $(d, ..., i-1, j, j+1, i+2, ..., d)$ and $(d, ..., j-1, i, i+1, j+2, ..., d)$.

**The Arc Relocation operator.** A segment of a route, consisting of at most $k$ vertices, is moved from one of the routes and inserted in another. An example of the arc relocation is seen on Figure 5.9. The arc relocation operator corresponds to the Or-opt operator in the multi route case.

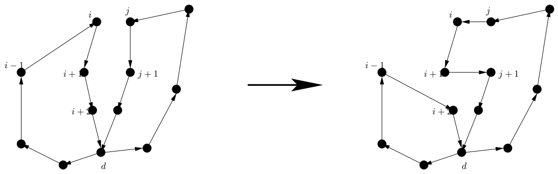


Figure 5.9: Example of the arc relocation, the route segment consisting of the vertices $i$ and $i+1$ and the arc $(i, i+1)$ is inserted between node $j$ and $j+1$. Such that the routes $(d, ..., i-1, i, i+1, i+2, ..., d)$ and $(d, ..., j, j+1, ..., d)$ change to the routes $(d, ..., i-1, i+2, ..., d)$ and $(d, ..., j, i, i+1, j+1, ..., d)$.

#### 5.2.6.1 The Chosen Neighbourhood

The implemented neighbourhood is known as the *Relocate operator*, it is described in various articles as [17]. The idea is to remove a customer from a position in a route, and reinsert it in another position in either the same or another route. The procedure is quite simple, but by repeating the procedure in different ways, the total improvement will be identical to that of other more complex improvement methods. When the time windows are narrow, other improvement methods like the arc exchange operator might easier reach part of the solution space, which is hard to reach with the relocate operator. Another quality of the relocate operator, is, that it can handle infeasibility, in that way, that customers which can not be serviced by the vehicles, due to time windows can be tried inserted in some of the partial constructed routes. This is of importance, since there are few vehicles to a lot of customers in the pack machine problem. The main reasons for choosing this neighbourhood, are that it is easy to understand, can yield the same improvements as other improvement methods, and the feature of handling infeasibility. The relocate operator is the special case of the String Relocation operator, where $k = 1$. An example of the chosen neighbourhood can be seen in Figure 5.10.

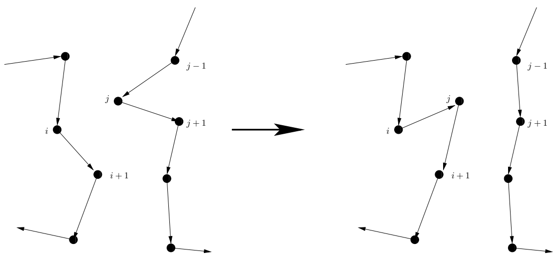


Figure 5.10: Example of the Relocate operator, vertex $j$ is removed from one route and inserted in another route between vertex $i$ and $i + 1$, such that the route segments $(i, i + 1)$ and $(j - 1, j, j + 1)$ are changed to $(i, j, i + 1)$ and $(j - 1, j + 1)$

In case of feasibility in the initially constructed solution, the moves supported by the neighbourhood are:

1. Removing a scheduled customer from one route to another route.
2. Removing a scheduled customer from one part of the route to another part of the route.

In case of infeasibility, the moves supported by the neighbourhood, are in addition to the ones described before, the following:

3. Removing a scheduled customer from the route, the removed customer will then be classified as an unrouted customer.

4. Inserting an unrouted customer in a partially constructed route.

This neighbourhood can be varied even more, according to how random the new solution should be. In the most random neighbourhood is the neighbourhood 1, 2, 3, or 4 chosen randomly. The customer to be removed, inserted or change route is chosen randomly. If the customer should be inserted in a new route, the route and the insertion place are chosen randomly. This neighbourhood is implemented, but the metaheuristic does not perform well using this neighbourhood. This indicates, that a less random neighbourhood might be more appropriate. One way to minimize the randomness of the neighbourhood is to insert customers in the route, such that the total length of the route is minimal, or investigate at which part of the route, the customer can be inserted according to time feasibility.

In this project the problem is assumed to be static. This is an approximation, since new customers can arrive during the campaign. When using move number 4, the neighbourhood would support a dynamic version of the problem. This is also the case for move number 3, if customers regrets their orders.

#### Implementing the chosen Neighbourhood

When using the chosen neighbourhood, one need to implement a procedure for inserting a customer in a partially constructed route, and one for removing a customer from a partially constructed route.

As seen in Section 5.2.5, the Push Forward operation can be used to update arrival times and checking for feasibility, when a customer is inserted in the route.

When removing a customer from a partially constructed route, the arrival times need to be updated too. There is however no need of a feasibility check, since the pack machines are allowed to arrive at a customer before the time window begins. In case of this, the pack machine just has to wait until the time window begins before serving the customers. The *Push Backward* operation, which is described in the next is used to update the arrival times, when a customer is removed from a route.

**The Push Backward Operation** The parameter $PB_{r,k}$, defines the Push Backward for customer $r$, when another customer $l$, scheduled before $r$ in the route $k$, is removed. It therefore defines how much the schedule is moved backwards in time, when a customer is removed, analogue to the Push Forward operation. It is worth noting, that if the route is time feasible, then removing a customer from the route can not make the route time infeasible. Therefore a time feasibility check is not needed here. The difference between the old and new arrival time at customer $l+1$ is given by $w_{l+1,k} - w^{new}_{l+1,k}$, this size defines how much the time schedule is *pushed backward* at customer $l+1$. Therefore the Push Backward operation at customer $l+1$ is given by (5.12).

$$PB_{l+1,k} = w_{l+1,k} - w^{new}_{l+1,k} \tag{5.12}$$

The difference between the old arrival time and the start of the time window for a customer $r+1$, indicates how much the time schedule can be pushed backward at maximum, according to the time windows for customer $r+1$. $PB^{max}_{r+1,k}$ can therefore be defined as (5.13).

$$PB^{max}_{r+1} = \max(0, w_{r+1} - a_{r+1}) \tag{5.13}$$

Since all the distances are euclidean, the Push Backward can never be less than zero. There are three instances, which leads to different calculation of the Push Backward. These are described in the next.

1. If $w_{r+1} \leq a_{r+1}$, then the time schedule can not be moved further backward, since it is the time window which decides the arrival time, therefore $PB_{r+1} = 0$.
2. If $w_{r+1} - PB_r \geq a_{r+1}$, then the arrival time for customer $r+1$ can be pushed backward as much as it was for customer $r$, therefore $PB_{r+1} = PB_r$.
3. If $w_{r+1} - +PB_r < e_{r+1}$, then the arrival time will be pushed back to the start of the time window. Therefore $PB_{r+1} = PB^{max}_{r+1,k}$

This leads to the recursive formula seen in (5.14).

$$PB_{r+1} = \min(PB_r, PB^{max}_{r+1}) \tag{5.14}$$

The complexity of the Push Backward operation is the same as for the Push-Forward operation, that is $\mathcal{O}(n)$. A move supported by the neighbourhood has therefore a complexity of $\mathcal{O}(n)$. The complexity increases, as the randomness of the neighbourhood decreases. The complexity of neighbourhoods with the different levels of randomness is described in the following, where it is assumed, that the customers are almost equally distributed in the routes. There are three moves with different level of randomness.

1. If the move is chosen completely random, that is, a random customer is tried inserted at a random route, in a random position, the complexity of the move is $\mathcal{O}(n)$.

2. In case of limited randomness, that is, a random customer is inserted in the route and at the position, which yields the best cost, the complexity will be $\mathcal{O}(n^2)$.

3. When no randomness is involved, and the best move in the neighbourhood is performed, the complexity of the move is $\mathcal{O}(n^3)$.

The two moves which include some randomness, can be used in Simulated Annealing, though the one with limited randomness, performs best. The complexity for an iteration in Simulated Annealing is when using limited randomness $\mathcal{O}(n^2)$. In the Steepest Descent algorithm, the best move in the neighbourhood is used, and therefore the complexity of an iteration will be $\mathcal{O}(n^3)$.

### 5.2.7 Evaluating the Solution

When a candidate solution is found in the neighbourhood, it has to be evaluated, such that it can be compared to the chosen solution. The solution can be evaluated by using the cost function of the original problem. But this will not be adequate, when the solution is infeasible. The cost could then be minimized when removing customers from the routes. Therefore a measure of how infeasible the solution is, needs to be included in the cost function. The infeasibility of a solution could be measured as:

- The number of unrouted customers.
- The total demand for the unrouted customers.
- The fraction of unrouted customers in the solution.
- The fraction of the total demand for the unrouted customers in the solution.
- A measure of how far the unrouted customers are from one of the constructed routes.

The number of unrouted customers is used as a measure of how infeasible the solution is. So the cost function is defined as seen in (5.15).

$$\text{cost} = \text{total time used} + \alpha * (\text{number of unrouted customers}) \tag{5.15}$$

This cost function deals with the solutions obtained in an iteration for either the Steepest Descent method or Simulated Annealing, and is not the same as the cost function stated in Equation (5.5) described in Section 5.2.3

### 5.2.8 Stopping Criteria for the Metaheuristic

One of the main disadvantages by using a metaheuristic such as Simulated Annealing is, that it is not known if the optimal solution has been found. Therefore it is not known, at which time the metaheuristic should terminate. The Steepest Descent method terminates, when no solution in the neighbourhood gives rise to an improvement in the cost function.

A stopping criterion is used to decide at which point the metaheuristic should terminate. The following criteria are the most used for Simulated Annealing.

1. If no improvements are achieved during the last $z$ iterations, the metaheuristic is stopped. The number $z$ is specified by the user.
2. A variant of the stopping criterion described in 1. Instead of looking at the absolute improvement, the focus is here on the relative improvement. Such that the metaheuristic should terminate, when the solution is not improved more than $x\%$ during the last $z$ iterations. The user should then specify $x\%$ and $z$.
3. The metaheuristic Simulated Annealing is stopped, when the temperature reaches a certain level defined by the user.
4. The metaheuristic is stopped, when the user is interrupting it.
5. The metaheuristic is stopped, after $z$ iterations, where $z$ again is specified by the user. This is actually the same as stopping criterion 3. But it is not dependant on the initial temperature $T$, the reduction factor $r$ or $L$, and it can then be used as a righteous stopping criterion, when different combinations of these parameters are tested, such that the algorithm terminates after the same number of iteration no matter which values the parameters take.
6. The metaheuristic is stopped after a given amount of time. This neighbourhood could be used, if the solution should be obtained according to given deadlines. This stopping criterion is a variant of 3 and 5, as the time is correlated with the number of iterations used and therefore also with the level of the temperature.

There is always a risk of stopping the metaheuristic just before big improvements are achieved. But some of the stopping criteria seems to be more likely to do this than other. When using the three last stopping criteria, one could risk to stop the metaheuristic, while the solution is improved. This is taken care of in criteria 1. and 2. On the other hand, these could run for a long time, where only small improvements are gained. The stopping criteria can be combined, such that the metaheuristic terminates when one of the chosen stopping criteria is reached.

In this project stopping criterion number 5 is used. The reason for choosing this stopping criterion, even though it has the disadvantage of not investigating whether the algorithm is in an improving phase, is that the comparison between the different sets of parameters is righteous.

## 5.3 Solving the Limited Pack Machine Problem

The limited pack machine problem were first known very late in the process, that is 5 weeks before hand in. Hence it was too late, to develop methods for solving this problem. Instead methods developed for the pack machine problem are used to solve the limited pack machine. In this section it is described how these solution methods can be used to solve the limited pack machine problem.

The limited pack machine problem will be solved with the same solution methods as used in the VRPTW approach. These are the nearest neighbour construction heuristic and the insertion heuristic for obtaining the initial solution, and the improvement heuristics Steepest Descent and Simulated Annealing.

The set of customers is initially divided in two set containing the customers in Scania and the rest of Sweden respectively. How the TSPs are solved for the two sets are described in Section 5.3.1 and 5.3.2

### 5.3.1 The Customers in Scania

The customers in Scania are initially divided in two sets corresponding to the two parts of the campaign. The TSPTW for the customers in the first part of the campaign is solved with the nearest neighbour heuristic and eventually the insertion heuristic. The time windows for the customers are identical, hence the problem could be solved with the methods described in Section 5.1.3. The route is improved with either Steepest Descent or Simulated Annealing.

The route containing customers which should be serviced in the second part of the campaign is constructed in the following way: The customers which should be serviced twice are initially inserted in the route in the same order as they were serviced in the first part of the campaign. The remaining customers are inserted with the insertion heuristic described in Section 5.2.4. The two routes are connected and the resulting route is improved with either Simulated Annealing or Steepest Descent.

### 5.3.2 The Customers in the Rest of Sweden

An initial solution is obtained with the nearest neighbour construction heuristic presented in Algorithm 6. As only one pack machine is available, only one route is constructed. In case of infeasibilty caused by not all customers are scheduled, the route is improved with the insertion heuristic described in Section 5.2.4. Finally Simulated Annealing or Steepest Descent is used to improve the initial solution.

Chapter 6

# Implementation

In this section the implementation of the solution methods used in this project is described. The implementation is realised in the programming language Java. The overall structure of the implemented program is described in Section 6.1. Some minor programs are made in addition to the big program, these are described briefly in Section 6.2 The source code for the programs can be seen in Appendix C.

## 6.1 Structure of the Program

The program is divided into several classes. The classes can be divided into three types of classes, those used to represent the customers, vehicles and routes, known as the data structure classes. The solution method classes builds the solution according to the methods described in Chapter 5. The last class type only consists of the class which reads the file, and transforms the data to the right structure. The connection between the classes can be seen in Figure 6.1.

In Section 6.1.1, the data structures used to model the customers, the routes and the pack machines are described. The implementation of the solution methods are described in Section 6.1.2 and in Section 6.1.3, for the VRP and the

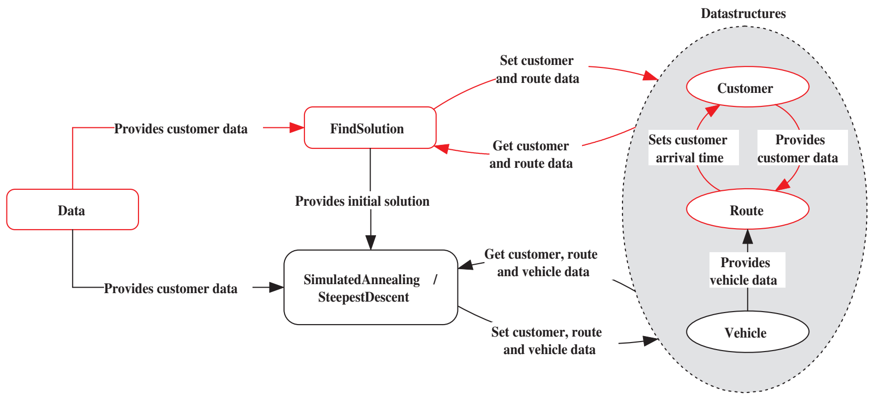



Figure 6.1: Connection between the classes in the implementation in VRP and VRPTW. The implementation for the VRP is marked with red.

VRPTW formulation respectively, while the read procedure and initial data representation is described in Section 6.1.4.

## 6.1.1 Data Structures

The data structures describe how the different elements in the problem are stored and implemented in the program. The data structures are almost identical in the VRP approach and the VRPTW approach of the problem. The differences between the data structures in the two approaches will be commented in the following paragraphs, where the classes are described.

### 6.1.1.1 The Customers

The customers are represented as two doubles representing their UTM coordinates, and a double representing their demand, in the VRP approach. In the VRPTW approach a customer is furthermore represented with a double array representing the time windows, and a double representing the arrival time for the customer, according to the schedule obtained by some of the solution methods. The implementation of the customers can be seen in Appendix C.1.1 and C.2.1.

#### 6.1.1.2 The Routes

The routes are represented as a vector containing the customers. The most important functions for the route in the VRP approach are listed in the next:

*addCustomer* appends a customer at the end of the route. By using the method *addCustomerAt*, the customer is inserted at the specified position of the route.

*removeCustomer* removes the given customer from the route.

*getCustomer* returns the customer.

*pathLength* returns the length of the path.

*routeSize* returns the number of customers in the route.

When using the functions *addCustomer*, *addCustomerAt* and *removeCustomer* in the VRPTW approach, the time is updated for the customers in the affected routes. The functions *pushForward* and *pushBackward* are used to update the times according to the procedures described in Section 5.2.5 and 5.2.6.1. The function *isInsertionTimeFeasible* uses the *pushForward* function to investigate, whether the route will remain feasible according to the time windows after inserting a new customer.

A solution is represented as $k = 3$ routes. The solution is feasible, if all customers are included in the routes, and the time windows are fulfilled for all customers.

#### 6.1.1.3 The Pack Machines

The pack machines are only represented as a data structure in the program of the VRPTW approach. The pack machines are represented by two doubles, the first one corresponds to the effect, that is the amount of bulk packed per hour, the second one corresponds to the number of working hours per day.

### 6.1.2 Solution Methods for the VRP Formulation

The implementation of the solution methods for the VRP formulation are implemented in one file, which can be seen in Appendix C.1. The implementation of the Sweep-algorithm and the Two-opt procedure, which is described in Section 5.1.2, is briefly commented in the next paragraph. The description only concerns the customers in the northern par of Sweden, since it does not make sense to apply this formulation and the method in southern and central part of Sweden, where the problem with double visits occur.

In Section 8.1 it is however investigated, what happens if the procedure is used to solve a problem, only consisting of customers, which should be serviced twice.

#### 6.1.2.1 The Two-phase Method

The customers situated in the northern part of Sweden is identified by the function *findNorthernCustomers*. These are divided according to their eastern coordinates, as described in Section 5.1.2.1. The TSP for each set of customers, is solved by the nearest neighbour heuristic with the function *nearestNeighbourNorth*. The resulting TSP solutions are improved with the 2-opt procedure by the function *twoOpt*, which uses the *twoExchange* function to perform the 2-exchange moves.

### 6.1.3 Solution Methods for the VRPTW Formulation

The solution methods for the VRPTW approach are described in the next paragraphs. In the first paragraph, it is described how the initial solution is obtained, the method for obtaining the initial solution, is the same for both Simulated Annealing and Steepest Descent. In the next paragraph the implementation of the metaheuristic Simulated Annealing is described, and finally, the implementation of the Steepest Descent improvement method is described in the last paragraph.

#### 6.1.3.1 The Initial Solution

The initial solution is obtained by the *nearestNeighbour* function, which is the implementation of the method described in Section 5.2.3. The function $costTW$ is used to evaluate the weighted cost of appending a customer at the end of a partial constructed route. When the nearest neighbour procedure has terminated, it is investigated whether there are customer, which are not scheduled in the solution. If that is the case, these customers are tried inserted in the solution by an insertion procedure, as described in Section 5.2.4.

#### 6.1.3.2 Simulated Annealing

The function *SA* is the implementation of the metaheuristic Simulated Annealing. It takes the initial solution, found by the methods described in the previous paragraph, as input. The initial temperature $T$, the reduction factor $r$, and the number of iterations between each reduction of $T$ are also inputs, which can be specified by the user. The methods for removing and inserting customers in the route, which are implemented in the *Route*-class, are used in the *SA* method, when finding new solutions in the neighbourhood. The cost of a given solution is evaluated by the function *costDistance*, when minimizing the total geographical route length is the objective. The total amount of time used by the pack machine can be calculated by the function *costTime*. The source code for Simulated Annealing can be seen in Appendix C.2.2.

#### 6.1.3.3 Steepest Descent

The solution method Steepest Descent, is implemented in the function *StD* in Appendix C.2.2. It uses the same functions in the *Route*-class as Simulated Annealing, for inserting and removing the customers from the routes. It uses the same initial solution, and the same functions for calculating the cost of a solution. The user only specifies, which solution should be used as initial solution.

### 6.1.4 Read Procedure

The implementation of the read procedure, can be seen in Appendix C.3. The function *readAndParseFile* takes the data file as argument, and reads the file. Some of the data has to be converted, this includes the coordinates and the time windows. The coordinates are converted to UTM coordinates by the function *geoToUtm*. The handwritten notes concerning the delivery dates according to the old schedule, is used to estimate the narrow time windows of $+/-5$ days in the VRPTW approach, while the *dlv. date* is used to calculate the timewindows for the limited pack machine problem. Each line in the data file corresponds to one customer. The customers are initialised, and stored in a vector.

## 6.2 Other Programs

In addition to the mentioned programs, some other types of programs are implemented. Some are implemented in Matlab, and used to visualize the solutions, others are made to generate data sets used in the parameter tuning described in Chapter 7. These programs are described in the next paragraphs.

#### 6.2.0.1 Matlab Files

Some minor matlab programs are generated by the Java programs. When one of the procedures terminates, the scheduled routes are plotted. For the solution methods as Simulated Annealing and steepest descent, the cost function is printed in each iteration, such that the development of the cost function can be plotted, when the method terminates.

#### 6.2.0.2 Generating Data Sets

The source code for generating data sets can be seen in Appendix C.4. The procedure for reading and storing the original data, is the same as described in Section 6.1.4. When the customers are stored in a vector, two functions can be used to generate other data sets. The function *reduceData* reduces the number of customers to a level specified by the user. Customers chosen randomly are removed from the route, until the specified level is reached. The function *changeDemand* changes the demands of the customers, such that the total amount of demand is unchanged. This is done by interchanging demands between the customers.

Chapter 7

# Parameter Tuning

For the solution methods used in the VRPTW approach, some parameters are defined to guide the performance of the solution methods. These parameters should be chosen, such that the best solution is obtained. In particular such parameters are used in the nearest neighbour construction heuristic, and the metaheuristic Simulated Annealing.

In this chapter, it is described how the parameters are tuned to perform the best as possible. The parameters should be tested and tuned on different data sets, than the one they are evaluated on, when verifying the quality of the method. When only testing on the evaluation data set, the parameters will be tuned to perform really good on this data set, while there is a great risk of performing bad on other data sets.

In this particular problem, only one data set has been supplied by Danisco. The data set has a special structure. The objective for the solution method, is to perform good, only on this particular problem, since the data do not variate much from year to year. The discussion on whether only to tune parameters on the given data set or on data sets with other structures is hard to decide. Another approach, where test data sets are formed from the original data set, is used. It can still be discussed, whether this approach is adequate, but it is chosen as a golden mean.

In Section 7.1, it is described how test data is obtained from the original data. In Section 7.2.1 and Section 7.2.2 it is presented, how the parameter tuning was performed for the methods nearest neighbour and Simulated annealing respectively. The results of the parameter tuning are presented in Section 7.3. The results are treated in this chapter instead of Chapter 8 to keep all concepts of the parameter tuning in the same chapter. A conclusion of the parameter tuning is presented in Section 7.4.

## 7.1 Obtaining Test Data

The test data sets are obtained from the original data set by introducing one of the following modifications.

1. Reducing the data, that is removing a specified number of customers from the data set. The removed customers are chosen randomly.
2. Changing demands, that is changing the demand for the individual customers, while the total demand stays the same. This is done by interchanging the demands between the customers. The demands are interchanged in a random manner.

The data sets used for parameter tuning, are presented in Table 7.1.

| name | size | method |
|---|---|---|
| `p370a.txt` | 370 | data reduced |
| `p370b.txt` | 370 | data reduced |
| `p420a.txt` | 420 | data reduced |
| `p420b.txt` | 420 | data reduced |
| `p470a.txt` | 470 | data reduced |
| `p470b.txt` | 470 | data reduced |
| `p570.txt` | 570 | demand changed |

Table 7.1: Structure of test data

## 7.2 Performing the Parameter Tuning

In this section it is described how, the parameter tuning was performed for the nearest neighbour construction heuristic, and Simulated Annealing. The parameter tuning is performed on the data sets described in Section 7.1. The two methods are tuned independently. It could be argued, that they should be tuned together, such that the best set of parameters, is obtained for the overall method. This would be a comprehensive task, as there would be many ways the parameters could be weighted in. It is assumed, that a good initial solution, will not reduce the performance of the metaheuristic.

### 7.2.1 The Nearest Neighbour Construction Heuristic

There are three parameters, which should be tuned for the nearest neighbour construction heuristic, these are $\delta_1$, $\delta_2$ and $\delta_3$ which are used in the weighted cost function in Equation (5.5). As stated in Equation (5.6) and (5.7), the parameters can not take negative values, and they should furthermore sum to 1. The parameters are normalized in the implementation of the function, as $\delta_1^* = \delta_1/(\delta_1+\delta_2+\delta_3)$, such that $\delta_1^*+\delta_2^*+\delta_3^* = 1$. The unnormalized parameters are taking the values as described in Table 7.2.

| parameter | value |
|---|---|
| $\delta_1$ | 0 1 2 3 4 5 6 7 8 9 10 |
| $\delta_2$ | 0 1 2 3 4 5 6 7 8 9 10 |
| $\delta_3$ | 0 1 2 3 4 5 6 7 8 9 10 |

Table 7.2: Structure of test data

All combinations of parameter values are investigated, except the instance, where all parameters takes the value 0. It is noted, that different combinations of parameter values, can yield the same set of parameters, when they are normalized.

### 7.2.2 Simulated Annealing

In the metaheuristic Simulated Annealing, the following parameters are tuned: the initial temperature $T_0$, the reduction factor $r$ and the number of iterations between each reduction of the temperature, $L$. In the parameter tuning described in this chapter parameters took the values described in Table 7.3. The objective is to minimize the total distance travelled, when performing the parameter tuning. Another objective could be to minimize the total amount of time used. This is however not carried out, but could be a relevant subject for further work.

| parameter | value |
|---|---|
| $T_0$ | 15 20 25 30 35 |
| $r$ | 0.88 0.90 0.92 0.94 0.96 0.98 |
| $L$ | 30 40 50 60 70 |

Table 7.3: Parameter tuning for Simulated Annealing

The range the parameters were tuned in, was chosen on the basis of some minor experiments on the original data, as no literature was found on this subject. The randomness of choosing a new solution in the neighbourhood leads to different solutions, hence a bad solution could be obtained, even though the parameters takes appropriate values. To reduce this, the procedure is performed 5 times for each combination of the parameter values, and the mean of the cost for these 5 runs is used to evaluate the performance of Simulated Annealing with these parameter values. For all data sets and all combinations of parameter values, the Simulated Annealing procedure was terminated after 7000 iterations. It does not count as an iteration, if the solution found in the neighbourhood is not time feasible. If the candidate solution is worse than the current solution and not accepted, it is still an iteration.

## 7.3 Results of the Parameter Tuning

### The Nearest Neighbour Construction Heuristic

In Figure 7.1 the results of the parameter tuning on the data set p370a is illustrated. The colours represent the cost of the solution, in this case, the cost corresponds to the number of customers which are not scheduled by the heuristic. Black represents the lowest cost, followed by the colours blue, cyan, green, yellow and red in increasing order, where the latter represents the worst solutions according to the cost.

From Figure 7.1 it is seen, that the best solutions is found, when $\delta_3$ is zero, when using the data set p370a. The same pattern is seen in the other test data sets, as presented in Appendix B.1.1.

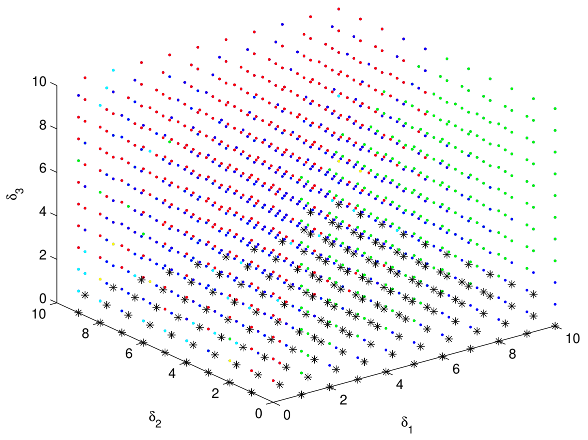



Figure 7.1: Parameter tuning for nearest neighbour construction heuristic on the data set p370a.

## Simulated Annealing

The results of the parameter tuning on Simulated Annealing is illustrated in the same way as the results of the parameter tuning for the nearest neighbour construction heuristic. The results for the data set p470b and p420a, can be seen on Figure 7.2 and Figure 7.3 respectively. From Figure 7.2, it is seen that the best averages of the solutions is obtained, when $T_0$ is low for the data set p470b, even though the best average, marked with black, of the solution is actually found for $T_0 = 35$. The same pattern is seen for the data sets p370b, p420b, p470a and p570, except that the actual best averages of a solution is not found for $T_0 = 35$. The results for the dataset p370b, p420b, p470a and p570 can be seen in Appendix B.1.2 on Figure B.1.2. No clear pattern is recognized for the other parameters, though it seems like the method performs better when $r$ is high.

It is more difficult to see a pattern from the results presented in Figure 7.3, the same is the case for the data set p370a, which is presented in Appendix B.1.2 on Figure B.4. Hence no conclusion can be made on these data set.

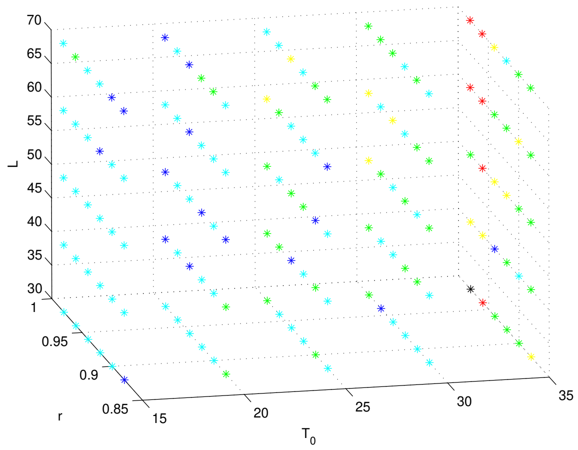



Figure 7.2: Parameter tuning for Simulated Annealing on the data set p470b.

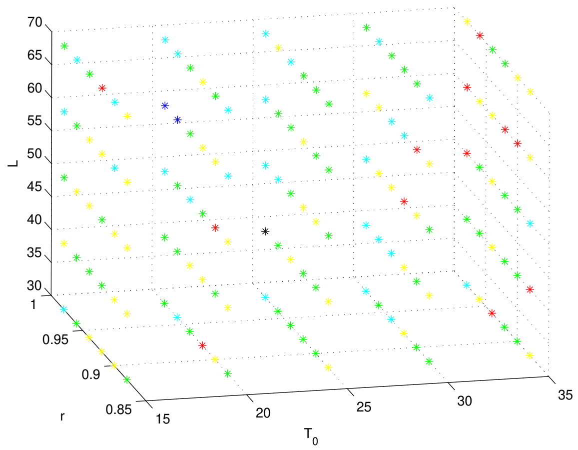



Figure 7.3: Parameter tuning for Simulated Annealing on the data set p420a.

## 7.4 Conclusion of the Parameter Tuning

It is seen, that the best solutions are obtained by the nearest neighbour construction heuristic, if the parameter $\delta_3$ is set to zero. Then it is not important, what value the parameters $\delta_1$ and $\delta_2$ takes. In this project $\delta_1$ is set to one, while $\delta_2$ and $\delta_3$ are set to 0. This is actual, the same as only using the geographical distance as a measure, of how far a customer is from the end of a route. The special structure of the time windows, which are made from a good solution, might cause this, as customers, which are situated close according to the geographical distance are often close to each other according to the temporal distance.

The results of Simulated Annealing are more difficult to interpret, though the results on most of the data sets indicate, that a low value of $T_0$ would be preferable. For Simulated Annealing the parameters are set as follows: $r = .98$, $T_0 = 15$ and $L = 50$.

Chapter 8

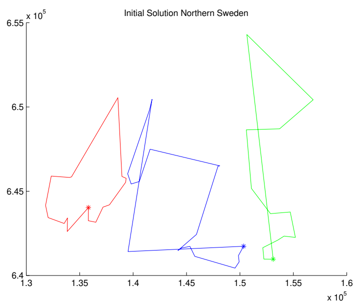
# Results

In this chapter, the results for the different approaches, i.e. the VRP, the VRPTW and the double TSP, are presented. In Section 8.1, the results for the VRP formulation are treated. Results for the VRPTW formulation are treated in Section 8.2. One of the obtaned solutions is compared with the solution provided by Danisco in Section 8.3. Finally the results for the double TSP formulation are treated in Section 8.4. The results concerning the parameter-tuning are found in Chapter 7. All experiments are performed on a SUN Fire 3800 computer with a 1200 Mhz processor.

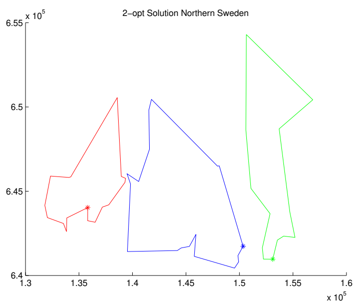
## 8.1 Results for VRP Formulation

The results of the VRP approach, are commented briefly in this section, as focus will be on the results of the VRPTW approach for the pack machine problem. Only the instance in the northern part of Sweden is considered. The customers belonging to the northern part of Sweden are here defined to be customers, with a northern UTM coordinate higher than 640000. There are 51 customers in this part of Sweden.

The customers are divided in three groups in the first phase of the two-phase heuristic. The TSPs in the different groups are solved by the nearest neighbour construction heuristic, as described in Section 5.1.2. The routes obtained by this procedure, are seen in Figure 8.1(a). The cost of the routes is defined as the travelled distance. The cost of the initial solution is 117 km. The computation time is 15ms. The routes are improved with the 2-opt procedure, as described in Section 5.1.3.2. The improved routes are depicted in Figure 8.1(b). The cost of the improved routes is 99 km. Hence the routes are improved 15,5% according to the cost function by the 2-opt procedure. The computation time for the 2-opt procedure is 867ms.




(a) Routes obtained by nearest neighbour




(b) Routes improved by the 2-opt procedure

Figure 8.1: Routes obtained by the two-phase heuristic with nearest neighbour construction heuristic. The routes are furthermore improved with the 2-opt procedure. The star indicates the initial customer.

It is seen, that the routes crosses their own paths in Figure 8.1(a), this is not the case on Figure 8.1(b), where the 2-opt procedure, has straightened the routes.

It is noted, that the results of the procedure depends on the customer, which is chosen as the initial customer in the construction heuristic. As the initial customer is chosen random, different solutions can be obtained, by performing the procedure several times. Hence the results treated in this section is just one instance of solutions obtained by the methods.

Furthermore it is noted, that the routes should not be closed for the pack machine problem. In Figure 8.1 the routes are closed, since this illustrates the 2-opt procedure better, as it is often the arc connecting the last customer with the initial customer, which causes bad solutions, this is seen for the green route, where this arc crosses the route 3 times in Figure 8.1(a).

**The Double Visits.** There is a problem concerning the customers which should be serviced twice. As expected, the second visit would be scheduled immediately after the first visit. An example of this can be seen, when the method is used on a set $\mathcal{C}_d$ containing only customers, which should be serviced twice. The routes will have the form $1_1, 1_2, ..., i_1, i_2, ..., n_1, n_2$, in the resulting schedule, when using the same notation as in Section 2.5.1. The routes from a solution obtained by this approach, can be seen in Appendix B.2.

## 8.2 Results for the VRPTW Formulation

The results for the VRPTW formulation include results of the initial solution obtained by nearest neighbour and the insertion heuristic, results of the Steepest Descent improvement heuristic and results of the metaheuristic Simulated Annealing. The results are presented in Section 8.2.1, 8.2.2 and 8.2.3 respectively.

### 8.2.1 The Initial Solution

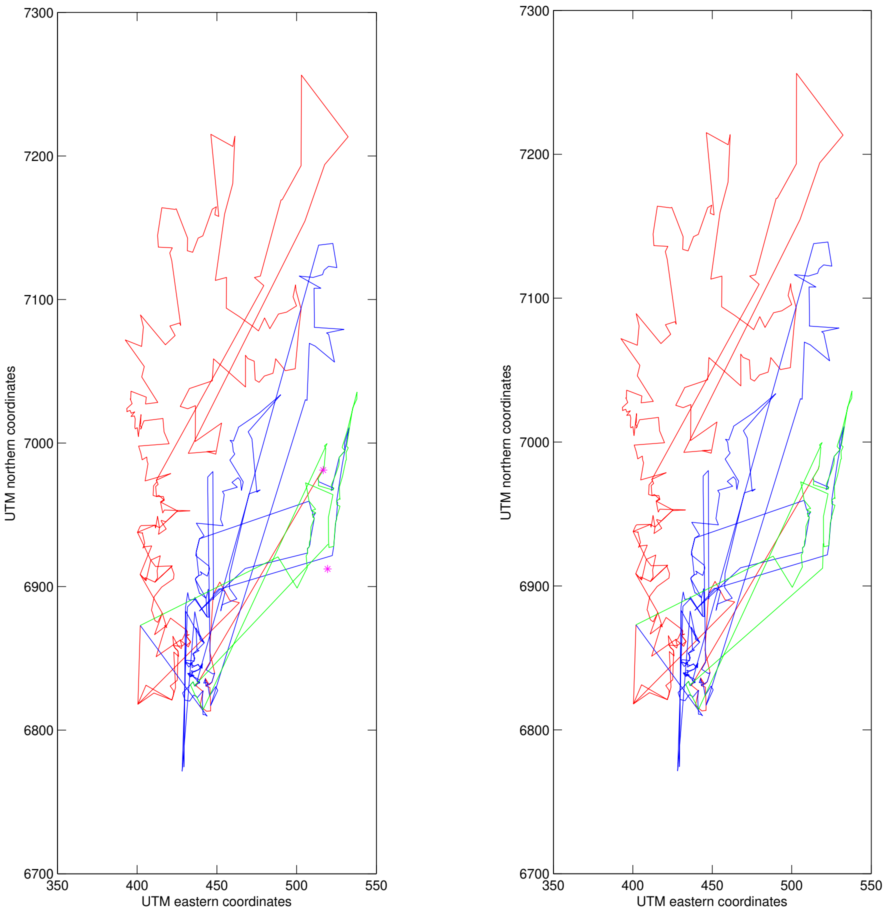
The parameters for the nearest neighbour construction heuristic are as a result of the parameter tuning chosen to be $\delta_1 = 1$, $\delta_2 = 0$ and $\delta_3 = 0$.

The initial solution obtained by the Nearest Neighbour construction heuristic is seen in Figure 8.2(a). The computation time for the nearest neighbour is 6933ms. There are 2 non-scheduled customers. The non-scheduled customers are marked with magenta stars in Figure 8.2(a). The computation time for the insertion procedure is 116ms.

By looking at the solution in Figure 8.2(a), it is seen, that some of the non-scheduled customers are located quite close to one of the routes in the partial solution. They might be missed because of the time windows, as they are not nearest neighbour when their time windows are open.

The infeasible solution is improved and made feasible with the insertion heuristic, where the 2 non-scheduled customers are inserted in one of the routes. The resulting solution is presented in Figure 8.2(b).

The cost of the solution could be interpreted as the total time used, not including waiting time or the total distance of the solution. The total time used is 190.1 days and the total distance travelled is 6451 km for the initial solution. The solution provided by Danisco has a cost of 199.6 days or 5226 km.




(a) Infeasible solution obtained by nearest neighbour.

(b) Improved feasible solution obtained by the insertion heuristic.

Figure 8.2: The infeasible initial solution, obtained by the nearest neighbour heuristic and improved by the insertion heuristic.

It was expected that the solution provided by Danisco had a better cost according to the travelled distance, as the greedy performance of the nearest neighbour construction heuristic could result in bad solutions. The fact that the obtained solution is better than the solution provided by Danisco according to the time used, might be caused by the way the routes are constructed. The pack machines have different effectivity, two of them have an effectivity on 50 tons per hour, whereas the last only produces 35 tons per hour. The routes are constructed sequentially, such that the two first routes, which is executed by the most effective pack machines are filled with as many customers as possible, while the last and less effective pack machine services the remaining customers. The last route contains the fewest number of customers, furthermore these customers are located far apart. This route is therefore characterized with a lot of transportation time compared to the amount of actual working time. As the transportation speed is the same for the three pack machines, such a route is actual preferable for the third pack machine in the solution, when minimization of time is the objective.

The problem concerning euclidean distances around *Øland* is seen for the routes, which services customers at the island, that is the green and the blue route. These routes switch between customers on *Øland* and customers on the mainland several times.

### 8.2.2 The Steepest Descent Method

In this section the results of the Steepest Descent improvement heuristic is presented. Two different solutions are used as initial solution for the method. That is the solution obtained by the nearest neighbour construction heuristic followed by the insertion heuristic, and the original solution provided by Danisco. Two different neighbourhoods are used:

*Limited neighbourhood.* The relocate operator is used, but only moves from one route to another is allowed.

*Unlimited neighbourhood.* The same neighbourhood as the limited, except that it is allowed to insert customers in the same route, they are removed from.

This leads to 4 different test scenarios:

1. Initial solution obtained by nearest neighbour construction heuristic and insertion heuristic. The neighbourhood is limited.

2. Initial solution obtained by nearest neighbour construction heuristic and insertion heuristic. The neighbourhood is unlimited.

3. Initial solution provided by Danisco. The neighbourhood is limited.

4. Initial solution provided by Danisco. The neighbourhood is unlimited.

### Minimization of time

The cost function is the same as defined in the VRPTW model in Equation (2.7), that is the total amount of time used in the solution not including the waiting time. The cost of the initial solution obtained by the nearest neighbour construction heuristic and improved with the insertion heuristic is 190.1 days, while the cost of the solution provided by Danisco is 199.6 days. The results of the test scenarios are presented in Table 8.1.

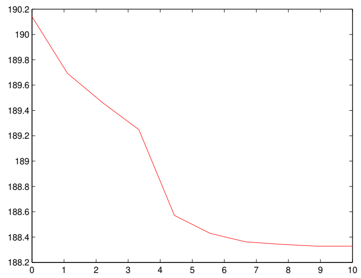
| Scenario | computation time | number of iterations | final cost | improvement |
|---|---|---|---|---|
| 1 | 877.620s | 10 | 188.3 | 0.9% |
| 2 | 4536.050s | 30 | 188.2 | 1.0% |
| 3 | 5528.040s | 54 | 186.8 | 6.4% |
| 4 | 17055.060s | 107 | 185.7 | 7.0% |

Table 8.1: Results of the Steepest Descent improvement method, when minimizing time is the objective.

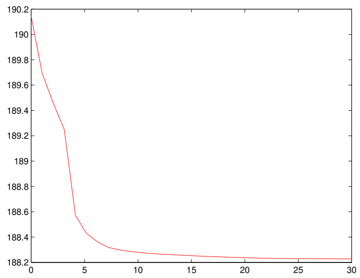
The evolution of the cost as function of the number of iterations used in Steepest Descent is depicted for the scenarios in Figure 8.3.

From Table 8.1 it is seen that using the unlimited neighbourhood increments the number of iterations and the computation time as well. The improvement gained by using the unlimited neighbourhood is small, 0.1 % and 0.6 % for the constructed solution and the original Danisco solution respectively, compared to using the limited neighbourhood. These observations are supported by Figure 8.3, where the slope of the cost is the same in the beginning of the process, i.e. the first 10 or 30 iterations for the obtained and provided solution respectively no matter what neighbourhood is used. Only limited improvements are obtained after the first 10 or 30 iterations as seen in Figure 8.3(b) and 8.3(d) respectively.

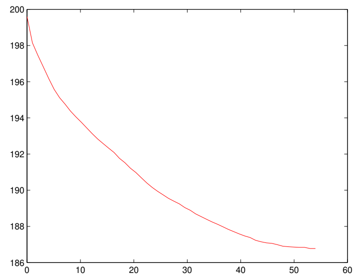
It is noted, that even though the solution provided by Danisco is worse than the obtained solution according to this cost function, the corresponding Steepest



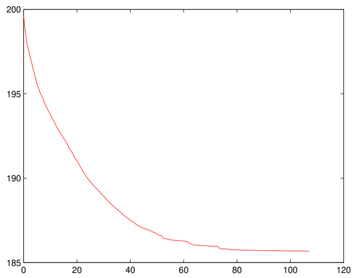

(a) Scenario 1




(b) Scenario 2




(c) Scenario 3




(d) Scenario 4

Figure 8.3: Cost as a function of the number of iterations performed with Steepest Descent for the 4 scenarios described in this section. Minimization of time is the objective

Descent solution is best. Hence it is seen, that a local minimum is reached when using the solution obtained by nearest neighbour and the insertion heuristic as the initial solution.

### Minimization of Distance

Experiments with the same 4 scenarios are performed with Steepest Descent, when minimization of distance is the objective. The results of the test scenarios are presented in Table 8.2.

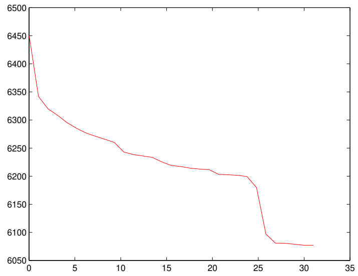
| Scenario | computation time | number of iterations | final cost | improve-ment |
|---|---|---|---|---|
| 1 | 3283.961s | 31 | 6077 | 5.8% |
| 2 | 7649.510s | 46 | 6148 | 4.7% |
| 3 | 1874.230s | 14 | 5115 | 2.1% |
| 4 | 20675.780s | 103 | 4741 | 9.3% |

Table 8.2: Results of the Steepest Descent improvement method, when minimizing the total distance is the objective.

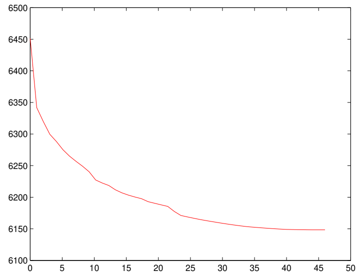
From Table 8.2 it is seen, that there are no great difference of using the limited or the unlimited neighbourhood, when using the solution obtained by nearest neighbour as the initial solution. According to the number of iterations and the improvement, it is noted, that the best solution actually is obtained with the limited neighbourhood. On the other hand the difference in computation time is big. For both neighbourhoods the algorithm terminates at a local optimum. The local optimum reached are for both neighbourhoods poor, as it is not even at the same level as the solution provided by Danisco. When using the solution provided by Danisco as initial solution, the results for the two methods differs more. When using the limited neighbourhood, few iterations are used, but the improvement is small. The contrary is seen, when using the unlimited neighbourhood. The computation time is long, but the improvement is impressive.

The cost functions are depicted for the four scenarios in Figure 8.4.

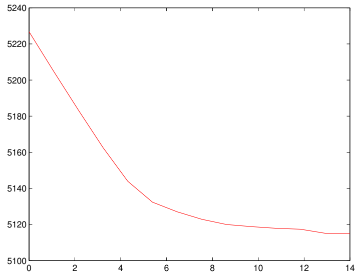
In Figure 8.4(a) it is seen that an impressive improvement is obtained at iteration 26. Without this move, the result of scenario 2 would have been better. It seems like a lucky coincidence, that a solution is reached where such a move is possible for the limited neighbourhood. As mentioned earlier, the number of iterations differs much between scenario 3 and 4, and the improvements differs much too. From Figure 8.4(d) it is seen that impressive improvements are gained in the



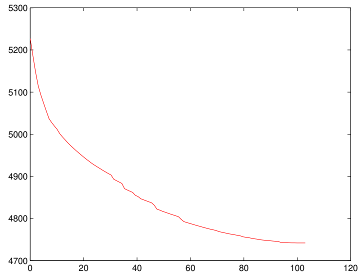

(a) Scenario 1




(b) Scenario 2




(c) Scenario 3




(d) Scenario 4

Figure 8.4: Cost as a function of the number of iterations performed with Steepest Descent for the 4 scenarios described in this section. Minimization of the total distance is the objective.

first part of the iteration process. The cost is after only 14 iterations much lower than it is when using the limited neighbourhood.

### 8.2.3 Simulated Annealing

The parameters for Simulated Annealing were tuned to minimize the distance, hence the results presented in this section are obtained with the objective of minimizing the total distance of the routes. The Simulated Annealing procedure has been performed 4 times with the parameters chosen on basis of the parameter tuning, the number of iterations is 7000 including *improvement moves*, *accepted* and *non accepted moves*. *Improvement moves* are moves where the current solution is improved, moves are *accepted* if they are accepted even though they provide a worse solution, otherwise the moves are *non accepted*. The computation time for Simulated Annealing using 7000 iterations is 1554.780s. The average result and the best of the results of the performances is presented in Table 8.3. The initial solution is the solution obtained by nearest neighbour, and the used neighbourhood is limited.

| Scenario | accepted and improvement moves | final cost | improve-ment |
|---|---:|---:|---:|
| Average | 195 | 6400 | 0.7% |
| Best | 131 | 6325 | 1.9% |

Table 8.3: Results of the Simulated Annealing metaheuristic, with the initial chosen parameters.

From Table 8.3 it is seen that the improvements obtained by Simulated Annealing are poor. The average improvement is lower than the lowest improvement for Steepest Descent. The best performance is slightly better, but still not impressive compared to Steepest Descent. This result is surprisingly, as it was expected that Simulated Annealing would perform better than Steepest Descent, because of the ability to avoid local optima. Another remarkable result is that the number of accepted moves and improvement moves are around 150, this is very low when the number of iterations are 7000. The development of the cost function is depicted on Figure 8.5, where the blue line shows the current cost and the red line shows the best cost found at a given time.

On Figure 8.5 it seems like the performance is still developing when the procedure is terminated. In the following it is investigated whether increasing the number of iterations would improve the performance. The results of increasing the number of iterations to 50000 is seen in Table 8.4, the value of the parame-

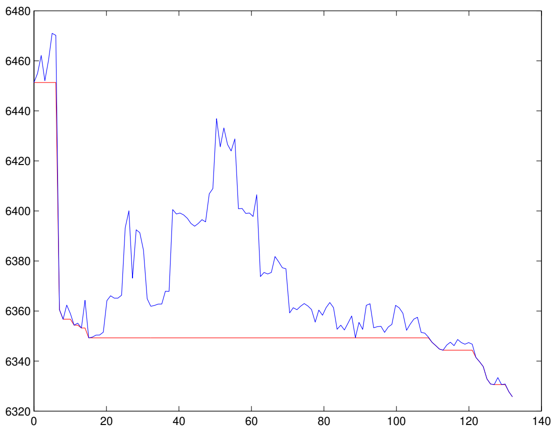



Figure 8.5: The cost as a function of the number of iterations performed with Simulated Annealing, using the parameters found in the parameter tuning.

ters, the initial solution, and the number of runs are the same as for the results in 8.3. The computation time is 12628,73s.

| Scenario | accepted and improvement moves | final cost | improvement |
|---|---|---|---|
| Average | 202 | 6392 | 0.9% |
| Best | 231 | 6372 | 1.2% |

Table 8.4: Results of the Simulated Annealing metaheuristic, with the initial chosen parameters and 50000 iterations.

From Table 8.4 it is seen, that the improvements are not greater though the number of iterations is increased to 50000. The number of improvement and accepted moves are just slightly higher at around 200. The number of accepted and improvement moves are extreme low according to the number of iterations. Hence increasing the number of iterations does not seem to improve the performance. The cost function for the best improvement is plotted in Figure 8.6, the evolution of the cost is quite similar to the evolution when only 7000 iterations are used.

As increasing the number of iterations does not seem to improve the performance of the metaheuristic, the cause of the bad performances must be due to something else, e.g. the value of the parameters. The initial temperature $T_0$ was set to 15, as it was indicated in the parameter tuning that a low value of

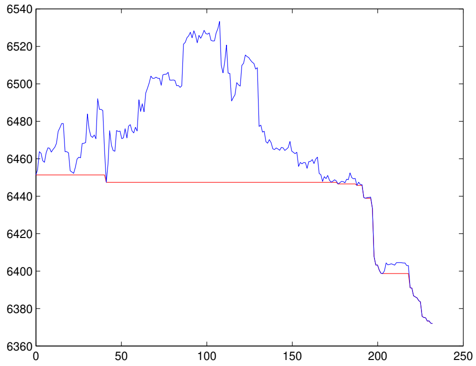



Figure 8.6: The cost as a function of the number of iterations performed with Simulated Annealing, using the parameters found in the parameter tuning and 50000 iterations.

$T_0$ yielded the best solutions. In the results however it was seen, that only a small part of the neighbourhood was used, as the number of moves were much lower than the number of iterations. According to [23] the *acceptance ratio* $\chi(T)$ could be used to tune the initial temperature $T_0$, the acceptance ratio is defined as the number of accepted moves divided by the number of proposed moves. According to [23] the acceptance ratio should be around 0.5 as a guideline. The acceptance ratio is in this project calculated from 500 iterations of Simulated Annealing with the initial temperature. The acceptance ratio were calculated to be 0.03 for $T_0 = 15$. This indicates that the initial temperature was too low. This seems strange as the most clear tendency in the parameter tuning was, that low temperatures yielded good solutions. When the temperature is low, Simulated Annealing process more like Steepest Descent, and this could cause the better solutions obtained when the temperature is low.

A new parameter tuning with higher temperatures could be performed, but as this is quite time consuming another approach is used. It is investigated whether an initial acceptance ratio at around 0.5 would improve the performance. The acceptance ratio is calculated for different values of the initial temperature. The result can be seen in Figure 8.7. It is seen, that the acceptance ratio which is closest 0.5 is obtained when the initial temperature is 85.

It is tested whether raising the initial temperature $T_0$ to 85, will improve the performance of Simulated Annealing. The computation time for running the performance with 7000 iterations is 1332.930s. But no improvements are seen.

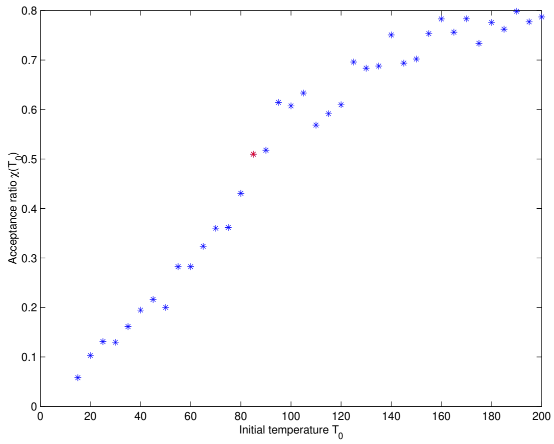



Figure 8.7: The acceptance ratio calculated for different values of $T_0$. A red star depicts the acceptance ratio which is closest to 0.5.

There are 436 accepted moves, which yields worse solutions, while the number of improvement moves is 434. The evolution of the cost function is seen on Figure 8.8.

From Figure 8.8 it is seen, that the cost is almost doubled in the first 250 moves. Afterward the solutions improves as the value of the cost decreases. The initial level of the cost is not reached during the computation, hence no improvement is seen. It is investigated whether increasing the number of iterations to 50000 will improve the performance, when using the initial temperature $T_0 = 85$. The resulting cost function can be seen on Figure 8.9.

The computation time for running Simulated Annealing with $T_0 = 85$ for 50000 iterations is 28714.790s, 515 moves were accepted while 555 moves were improvements moves. From Figure 8.9 is is seen, that the cost is in fact doubled during the 400 first moves. In the next 500 moves the value of the cost decreases and in the last 150 moves the cost function levels out, even though the cost function is still higher than the original cost, and also higher than the final cost of running the performance with 7000 iterations. The difference between the performance when using 7000 and 50000 iterations are caused by the randomness of Simulated Annealing.

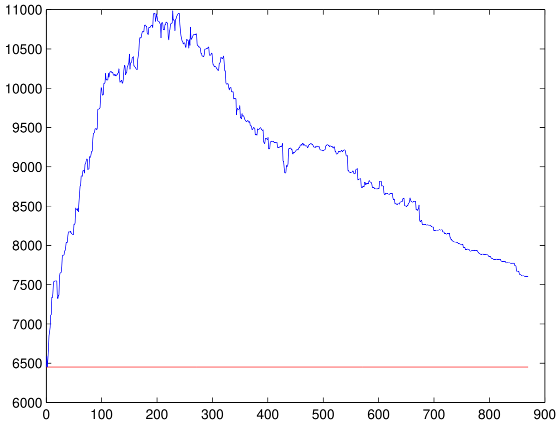



Figure 8.8: The cost as a function of the number of iterations performed with Simulated Annealing, using $T_0 = 85$ and 7000 iterations.

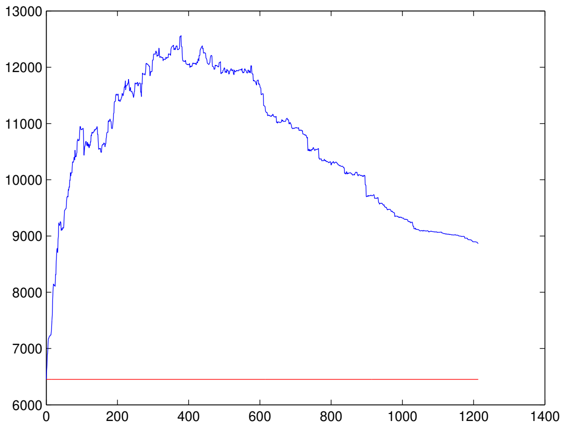



Figure 8.9: The cost as a function of the number of iterations performed with Simulated Annealing, using $T_0 = 85$ and 50000 iterations.

## 8.3 Comparing the Results with the Solution Provided by Danisco

In Figure 8.10 the solution provided by Danisco and the solution obtained by nearest neighbour and the insertion heuristic, and improved by Steepest Descent is seen. The solution is obtained with the objective of minimizing time.

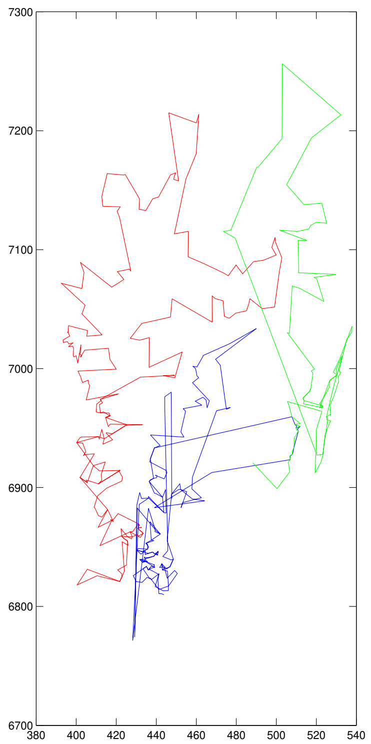
From Figure 8.10 it is noted, that both solutions seem improvable, as the routes crosses their own paths and the other routes several times. This could be caused by the time windows, but it is so pronounced, that it seems reasonable, that the routes could be improved further. It is noted, that the obtained solution seems worse, than the solution provided by Danisco.

The routes depicted with red lines are for a great part almost identical in the two solutions. They differ as some of the north-eastern most customers are visited by the red route in the solution obtained by Steepest Descent, and by the green route in the solution provided by Danisco. It is also seen, that some parts of the blue routes are identical, but in the solution obtained by Steepest Descent, more customers are visited by this route, that is mainly customers located in the eastern part of Sweden, customers which are serviced by the green route in the solution provided by Danisco. There are fewer customers, which are visited by the green route in the solution obtained by Steepest Descent, than in the solution provided by Danisco. The total distance is shorter for the solution provided by Danisco, whereas the total time used is lowest for the solution obtained by Steepest Descent, as the red and blue routes, which is driven by the most effective pack machines visits more customers in this solution.

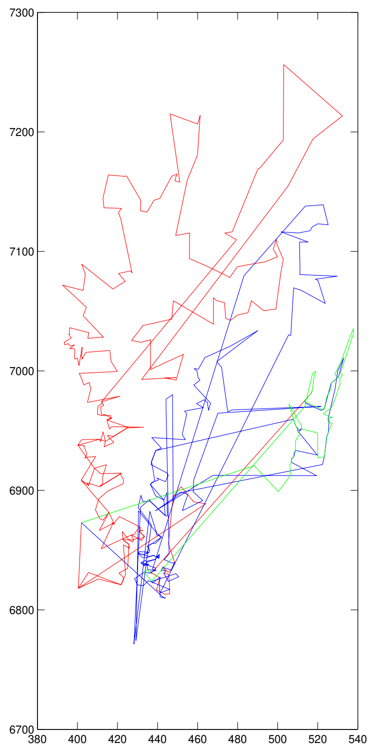
## 8.4 Results for the Limited Pack Machine Problem

The limited pack machine problem is initially solved with the construction heuristic nearest neighbour and the insertion heuristic. The initial solution could be improved by Steepest Descent or Simulated Annealing, but as the best results for the pack machine problem is obtained with Steepest Descent, only the performance of this method is evaluated for the limited pack machine problem. As only one pack machine is used for each of the customer sets, the unlimited neighbourhood is used in the experiments.

It is determined by Danisco that the most effective pack machines should service




(a) Solution provided by Danisco




(b) Solution obtained by Steepest Descent

Figure 8.10: The original solution provided by Danisco and the solution obtained by Steepest Descent when minimizing time, the units of the horizontal and the vertical axis are respectively the eastern and northern coordinates represented as UTM coordinates divided by 1000.

the customers in the two sets. Minimization of total distance is consistent with minimization of time. Minimization of distance is the only objective in the results described in the next sections. The results concerning the customers in Scania are presented in Section 8.4.1, while the results concerning the rest of the customers are presented in Section 8.4.2.

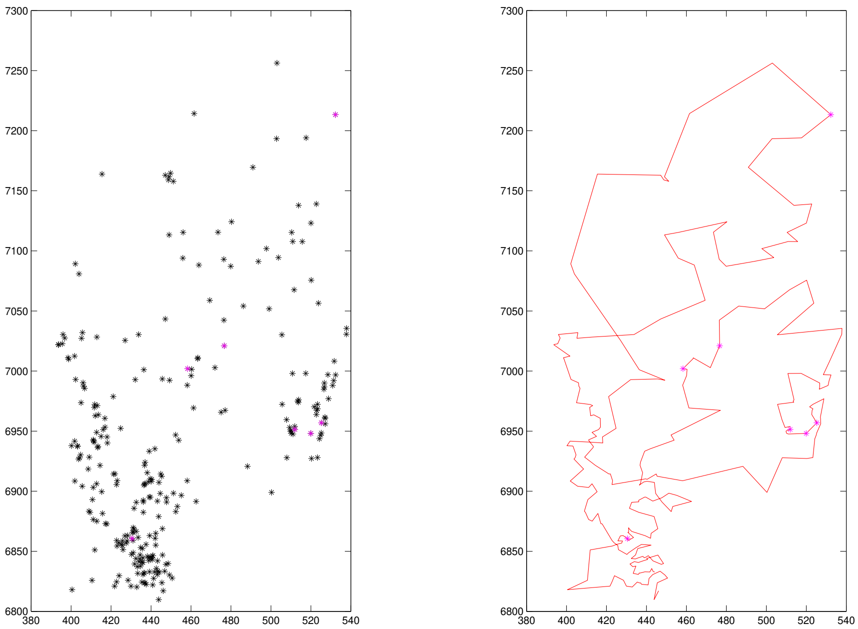
### 8.4.1 The Customers in Scania

It was initially not possible to obtain a feasible solution for the limited pack machine problem for the customers in Scania, that is customers with the geographic zone 401. It was observed, that all customers should be serviced in the first part of the campaign. The first part of the campaign consists of 60 days for the limited pack machine problem. The solution obtained had a duration of 65 days. It was also noted, that the customers which should be visited twice, demanded both visits in the first part of the campaign. These customers were identified, and one of the visits was moved to the second part of the campaign. There were only 8 customers which should be visited twice in the geographic zone 401. Hence only 8 customers should be visited in the last part of the campaign. In Figure 8.11(a) the customers with the geographic zone 401 are depicted. Customers which should be serviced twice are marked with magenta stars. It is noted, that the shape of Sweden is still seen on the figure. This is strange, as customers with geographic zone 401 should be located in Scania according to Danisco. Unfortunately, time did not allow further investigations of this.

As very few customers are visited in the second part of the campaign, the first part of the campaign is exceeded with 10 days, such that 70 days are available in this part of the campaign. Furthermore only customers in the first part of the campaign is considered when solving the problem. The resulting problem could be regarded as a TSP without the time windows, as all customers have the same time windows.

The computation time for solving this problem with nearest neighbour and Steepest Descent is 3310.900 seconds. The distance of the initial solution is 2480 and the distance for the improved solution is 2362, hence the solution is improved 4.8% with the Steepest Descent method. The obtained route is seen in Figure 8.11(b), where the magenta stars mark customers which should be visited twice. The evolution of the cost function is seen on Figure 8.12.

The obtained route depicted in Figure 8.11(b) is improvable, as the route crosses its own paths several times. Since the time windows were identical for all customers, the bad solution could not be explained by the time windows.




(a) Customers in Scania

(b) Obtained route

Figure 8.11: Route obtained with Steepest Descent for the limited pack machine problem in Scania.

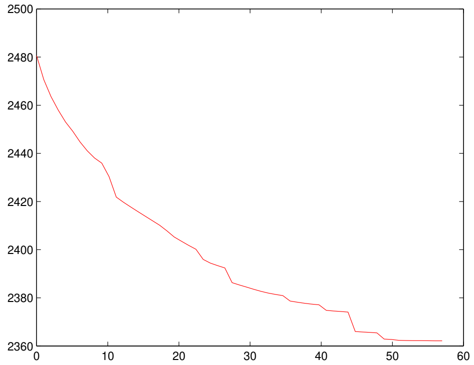



Figure 8.12: Cost as a function of the number of iterations performed with Steepest Descent for the limited pack machine problem in Scania.

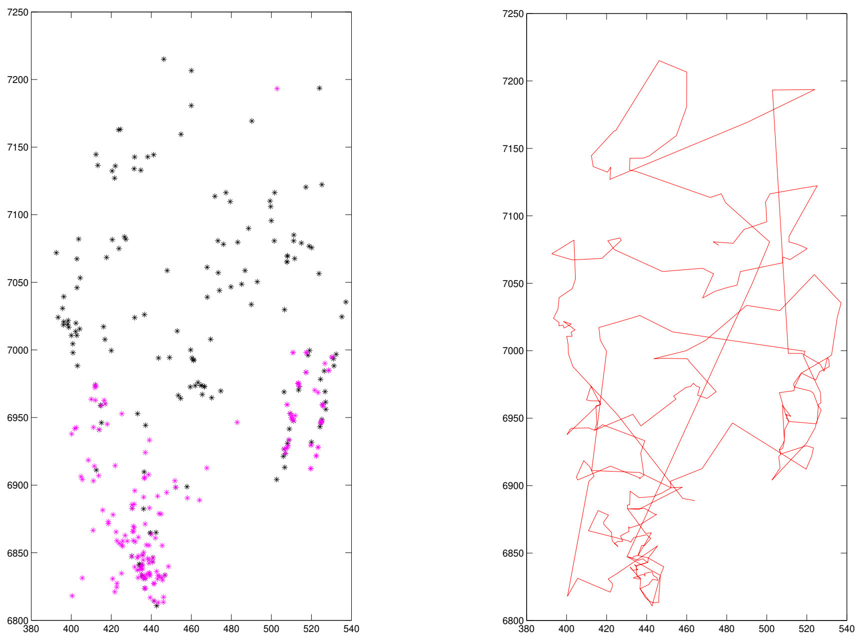
From Figure 8.12 it is seen, that the cost decreases most during the 30 first iterations, and levels slightly out during the last iterations.

### 8.4.2 The Customers in the Rest of Sweden

The customers located in the rest of Sweden are seen in Figure 8.13(a), these are the customers with other geographic zones than 401. Customers which are serviced in the last part of the campaign are depicted with magenta stars. The shape of Sweden is still seen, and it seems like the customers located in Scania are included in the set. Further investigation of this strange incident, was not carried out due to lack of time.

The TSPTW for the customers in the rest of Sweden is solved initially with the nearest neighbour and the insertion heuristic, and further improved with Steepest Descent. The cost of the initial solution is 3562 km, while the cost of the improved solution is 3504, hence an improvement 1.6% is obtained. The computation time is 380.190s. The improvement of the solution is limited, but the computation time is low too. It seems like a local optimum is reached after very few iterations. The obtained solution is seen in Figure 8.13(b) while the cost function of Steepest Descent is seen in Figure 8.14.

From Figure 8.13(b) it is seen, that the obtained route seems to have a poor quality, as the route crosses its own paths several times. As the time windows




(a) Customers in the rest of Sweden

(b) Obtained route

Figure 8.13: Route obtained with Steepest Descent for the limited pack machine problem in the rest of Sweden.

are wide this can not explain the poor quality of the solution. Instead the poor solution must be explained by the fact, that a very bad local optimum is reached.

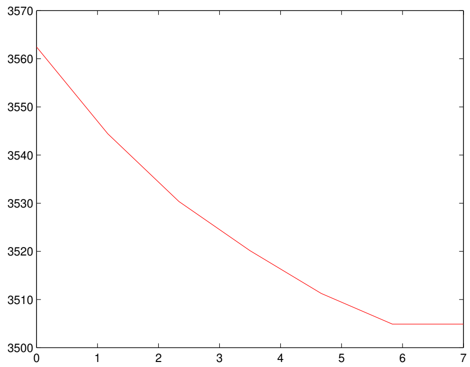



Figure 8.14: Cost as a function of the number of iterations performed with Steepest Descent for the limited pack machine problem in the rest of Sweden.

As seen in Figure 8.14 only 7 iterations of Steepest Descent is performed before the local optimum is reached. This was expected, as the computation time was low, and the solution was only improved slightly. The slope is almost constant in the first 6 iterations, whereas the improvment in the last iteration seems insignificant.

Chapter 9

# Exact methods

In Section 9.1 and 9.2 it is investigated how the problem could be solved to optimality. When solving real-life problems, it is often more appropriate to use a metaheuristic, since it is possible to terminate such methods, when a solution is needed. Exact methods are interesting in an academic point of view, since the optimal solution can be used to evaluate the performance of a metaheuristic. Like metaheuristics can be used in various types of optimization problems, the exact method *Branch & Bound* can be used to solve different kinds of optimization problems.

The theory behind *Branch & Bound* is presented in Section 9.1. How the method can be applied to the VRPTW formulation of the problem is described in 9.2.

## 9.1 Branch & Bound

The method Branch & Bound can be used to solve general integer problems. One method to find an optimal solution, is to investigate the solution space explicit. But when the solution space is big, this would be very time consuming. Branch & Bound uses a strategy, where the solution space is divided into smaller disjunctive solution spaces. By using knowledge of primal and dual bounds of the solution, upper and lower bounds in case of minimization, some of the smaller solution spaces are eliminated, and therefore the total solution space is minimized. The theory behind Branch & Bound is described further in [4].

### 9.1.1 Branching

The divided solution space, which is obtained, when using Branch & Bound, can be illustrated as a tree. The root of the tree corresponds to the original problem, the nodes under the root correspond to the problems which only contains a part of the solution space. The leafs correspond to solutions to the problem. At each level the solution space is divided into smaller solution spaces, such that the solution spaces corresponding to the children of an parent-node united will add up to the solution space of the parent. The solution space is divided by adding constraints to the parent node. Division of the solution space is for obvious reasons named branching.

### 9.1.2 Primal and Dual Bounds

The Branch & Bound algorithm, uses information of primal and dual bounds to reduce the solution space, that is to prune branches of the Branch & Bound Tree. The concept of primal and dual bounds is described in the next.

#### Primal Bounds

The primal bound is the same as an upper bound for a minimization problem. Every feasible solution to a problem will give a primal bound. The quality of a primal bound depends on how close it is to the optimal solution. For some problems, it is easy to find feasible solutions, and the solution can be found in polynomial time. This is though not the case for all problems, e.g. it is an $\mathcal{NP}$-hard problem to find a feasible solution to the VRPTW, when there are more customers than vehicles according to [5]. Heuristics are often used to obtain a feasible solution, which can yield the primal bound. The solution, which provides the best primal bound in the Branch & Bound algorithm, at a given time is called the incumbent.

### Dual bounds

The dual bound constitutes a lower bound for a minimization problem. That is, a value which is known to be at least as good or low as the cost of the optimal solution. The dual bounds are found by using either *relaxation* or *duality*. The definitions of relaxation and duality are stated in Definition 2 and 3.

**Definition 2** *A problem* $P^r = \max\{f(x) : x \in T \subseteq R^n\}$ *is a relaxation of the problem* $P = \max\{c(x) : x \in X \subseteq R^n\}$ *if:*

1. $X \subseteq T$ *and*
2. $f(x) \geq c(x)\ \forall\ x \in X$

Because of the second property, the optimal solution of a relaxed problem, will always provide a dual bound to the original problem.

**Definition 3** *The two problems* $P : z = \max\{c(x) : x \in X\}$ *and* $DP : w = \min\{\omega(u) : u \in U\}$ *are*

*weak dual if* $c(x) \leq \omega(u)$ *for all* $x \in X$ *and* $u \in U$ *and*

*strong dual if they are weak dual and in addition* $z = w$

The solution of a weak dual problem yields a dual bound for the original problem.

Some properties of relaxations and duality are stated in [4]. When the dual bound equals the primal bound, an optimal solution is found. Many optimization algorithms will tend to find new solutions, for which the dual bound increases and the primal bound decreases, during the execution of the algorithm, when minimization is the objective.

## 9.1.3 Pruning

As written in Section 9.1.2, the incumbent provides a primal bound for the solution value. Sometimes the incumbent can be found by a heuristic. If it is cumbersome, to find an incumbent in the solution space, one could branch until the solution space has a reasonable size. By using knowledge of the incumbent and the dual bounds, it is possible to discard some parts of the solution space, illustrated in the Branch & Bound-tree by *pruning* the branches. There are three cases, where pruning is possible.

1. The dual bound equals the primal, the solution is optimal according to the solution space for the branch and becomes a leaf in the B & B tree. This is known as pruning by optimality.

2. The solution space is empty, there is no solution found in this part of the solution space, this is known as pruning by infeasibility.

3. The dual bound is higher than the cost for the incumbent for a minimization problem. This is known as pruning by bounds.

The branch which is pruned will not be investigated further unless it is pruned by optimality. When no pruning is possible, the solution space for the node is divided, as described in Section 9.1.1.

When all branches in the tree are either pruned or proved to be optimal as leafs, the optimal solution is found, in the leaf, which has the lowest value, in case of minimization.

### 9.1.4 Searching Strategies

An *active node*, is a node in the B & B- tree which is not pruned yet. Active nodes have to be investigated further. Since there might be more than one active node, a strategy of how to choose which active node, that should be investigated next, is needed. Some of the most common search strategies for choosing the active node are listed in the next paragraphs.

**Best First Search Strategy (BeFS):** The active node which has the lowest dual bound (minimization) is investigated first. The idea behind BeFS, is that a good incumbent, will cause greater possibility for pruning by bound. One can risk, that the number of active nodes becomes large, when using BeFS. This causes a problem, when the computer has limited memory.

**Breath First Search Strategy (BFS):** All active nodes on a given level of a tree are investigated, before branching and continuing the procedure for the next level. This method is not appropriate, since the number of leafs grows exponentially, according to the level of the tree, and it will therefore cause memory problems too.

**Depth First Search Strategy (DFS):** A feasible solution which yields a good primal bound is essential for the Branch & Bound algorithm. Such solutions can be found by investigating leaves deep in the tree. When using DFS, the active node which is at the deepest level is investigated first. The number of active nodes is growing linearly with the number of levels in the tree in this strategy, and DFS is therefore not very memory consuming. If the solution found yields a bad primal bound, the number of neccesary calculations is high.

### 9.1.5 Preprocessing

It seems easier to solve problems, by using Branch & Bound if the solution space is small and described precisely. The concept behind preprocessing is to improve the formulation of the problem, such that a smaller solution space and a more precise formulation is obtained. Some of the most typical ways of improving the formulation are listed in the next paragraphs:

**Remove redundant constraints.** By using knowledge of the bounds of the variables, it can be seen, that some constraints will never be violated, when the bounds on the variables are fulfilled. These constraints are redundant, and can therefore be removed from the formulation.

**Tightening bounds.** By using knowledge from the constraints, bounds can be tightened for different variables.

**Variable fixing.** If a variable does not have influence on any constraint which involves other variables, it can be fixed at the value, which yields the best objective, according to the bounds.

**Discard unneccesary parts of the solution space.** If it is known, that the optimal solution can not be found in a part of the solution space, this part of the solution space can be discarded.

## 9.2 A B & B Algorithm Applied on VRPTW

The Branch & Bound method can be used to solve the VRPTW to optimality. In this section, it is described how the problem formulated as a VRPTW can be solved. In [18] a method to obtain a dual bound, based on column generation is described. In [5] the bound found by column generation is improved by using *2-path Cuts*. A branching strategy is proposed in [5]. Both methods are applied to the general VRPTW formulation. They can also be adapted to the special VRPTW formulation, used in this project.

### 9.2.1 Obtaining a Dual Bound

The dual bound is obtained by relaxing the VRPTW formulation. The constraint in (2.8) is relaxed linearly. The relaxed problem is solved with column generation, also known as Dantzig-Wolfe decomposition.

The main concept behind column generation, is to divide a problem into two easier solvable problems, which are solved several times, such that informations from solutions, are used to update either the solution space or the cost function. In the next, it is described, how the column generation can be applied to the model, described in Section 2.2. By introducing some minor modifications, the method can be applied to the other mathematical model, the VRP model presented in Section 2.1. This is not described, since this section is not the main part of the project. The two problems are called the *master problem* and the *subproblem*. By looking at the mathematical model described in Section 2.2, it is seen, that the constraint in (2.8) is the only constraint, which links the pack machines together. This constraint is also known as the semi-assignment constraint, since it states, that each customer must be assigned to precisely one pack machine. The rest of the constraints, these are constraint (2.9) to (2.16) constitute a *Shortest Path Problem with Time Windows* for each of the pack machines. In contradiction to the normal Shortest Path Problem, where only the edges has a cost, the customers which is represented as nodes has a cost too.

The idea of dividing the problem is used to define the *master* and the *subproblem*, which is described in the next.

**The Master Problem.** The objective for the VRPTW is to find a set of minimum cost routes, which start and end at a given depot $d$. The pack machines are servicing customers with a known amount of demand, which should be fulfilled. Every customer should be serviced exactly once. The service for customer $i$ begins at time $T_i$, which is inside the interval $[a_i, b_i]$, these time windows are hard, that is, they can not be violated. This can be formulated as a *Set Partitioning problem*, where:

$\mathcal{R}$ is a set of feasible routes for the VRPTW.

$\delta_{ir}$ is a constant which is:

1 if the route $r \in \mathcal{R}$ visits customer $i$ and

0 otherwise.

In the set partitioning problem, some routes are chosen, such that they make up a set of routes which adhere the VRPTW constraints, and yield a minimum cost. The decision variable $x_r$ takes the value 1, if the route $r$ is chosen in the solution, and 0 otherwise. The mathematical formulation of the Set Partitioning problem is described in Equation (9.1) to (9.3).

$$\min \sum_{r \in \mathcal{R}} c_r x_r \tag{9.1}$$

subject to

$$\sum_{r \in \mathcal{R}} \delta_{ir} x_r = 1 \qquad \forall \qquad i \in \mathcal{N} \setminus \{d\} \tag{9.2}$$

$$x_r \in \{0, 1\}, \quad r \in \mathcal{R} \tag{9.3}$$

The vector which contains the elements $\delta_{ir}$ $\forall$ $i \in \mathcal{N} \setminus \{d\}$ is called a column, and represents a feasible route. There are a lot of feasible routes, and it is very time consuming to find all feasible routes. Therefore the columns representing the routes are generated as they are needed, by using a column generation method. The subproblem is used to decide, which route should be added to the master problem. It is stated in [18], that the linear relaxation of a *Set Covering* model would be more numerical stable, than the set partitioning model presented in (9.1) to (9.3). The Set Covering model is obtained by exchanging the constraint given in (9.2), with the constraint given in in (9.4) as stated in [18].

$$\sum_{r \in \mathcal{R}} \delta_{ir} x_r \geq 1 \; \forall \, i \in \mathcal{N} \setminus \{d\} \tag{9.4}$$

It is seen, that the only difference between the set covering and the set partitioning models are the $\geq$-sign instead of the $=$-sign in the constraints.

**The Subproblem.** When the master problem as described in the previous paragraph is solved to optimality, it is not given, that the solution is optimal to the original VRPTW. This is caused by the fact, that there might be routes not included in $\mathcal{R}$ which can improve the solution. How to find and generate such a route, and to check whether there exist such routes, which are not contained in $\mathcal{R}$ is described in the next. The routes should be generated, such that they will improve the solution of the master problem. From the theory of duality it is known, that a new route in the master problem can improve the solution, if the reduced cost belonging to this route is strictly negative in the master problem [5]. The reduced cost $\bar{c_r}$ for route $r$ is given in Equation (9.5).

$$\bar{c_r} = c_r - \sum_i \pi_i \delta_{ir} \tag{9.5}$$

Where $\pi_i$ are the dual variables connected with the assignment constraint for customer $i$ in route $r$ given in (9.2) for the set partitioning model. The reduced cost as presented in (9.5) is for the total route. By using the rewriting in (9.6), it can be seen how the arcs influence the reduced cost for the route.

$$\begin{aligned}
\bar{c_r} &= c_r - \sum_i \pi_i \delta_{ir} \\
&= \sum_{ij} c_{ij} x_{ijr} - \sum_i \pi_i \delta_{ir} \\
&= \sum_{ij} c_{ij} x_{ijr} - \sum_i \pi_i \sum_j x_{ijr} \\
&= \sum_{ij} (c_{ij} - \pi_i) x_{ijr} \qquad (9.6)
\end{aligned}$$

The objective of the subproblem, is to find a route, where the total reduced cost is minimized. In addition the route constraints (2.9) to (2.16) in the VRPTW model should be fulfilled. By solving the subproblem to optimality, the route with most negative reduced cost is generated, and can be included in the set of routes in the master problem.

The subproblem could be solved to optimality by using an optimization solver as CPLEX. In [18] it is instead proposed to solve the subproblem by a dynamic programming algorithm, where not only the route with the most negative reduced cost is generated, but also other routes, with negative reduced cost. The problem will then be faster to solve, as less iterations are needed.

When solving the subproblem, it should be taken into account, that the pack machines have different effectivity. This influences the duration of he paths obtained. This problem could be solved by introducing different types of subproblems for the different types of pack machines.

The Column Generation procedure terminates, when there are no more negative reduced costs in the master problem. Since the master problem is linearly relaxed, an integer solution is not guaranteed. The solution of the column generation method will then provide a dual bound for the VRPTW problem. If the obtained solution is integer, it will be the optimal solution to the VRPTW problem.

This Column Generation method is applied to the VRPTW formulation and it can also be used in the VRP formulation if the time window constraints are omitted in the subproblem. It is stated in [18] that the set partitioning model is not adequate, when many customers are visited in the same route. This is indeed the case in the pack machine problem, where the number of vehicles are low compared to the number of customers. It will then be cumbersome, to apply the set partitioning model on the pack machine problem.

#### 9.2.1.1 Column Generation and Formulation of Double Visits

The problem concerning the customers which should be serviced twice, could be modelled as in Equation (2.24). If this formulation is used, the constraint should be incorporated in the master problem, since it can not be applied for one single route, as the customer can be visited by two different pack machines. The dual variables from this constraint should be added to the cost function in the subproblem.

### 9.2.2 Improving the Dual Bound

In general, the Branch & Bound procedure proceeds better, when the solution yielding the dual bound is closer to a feasible solution. In [5] a method for improving the dual bound obtained by column generation is derived. The method is based on the *Branch & Cut*. The idea is to identify a constraint, which should be fulfilled according to the problem formulation, but is not fulfilled in the solution yielding the dual bound. The problem of finding such a constraint is called a *separation problem*. The identified constraint is added as a cut to the problem formulation and the problem is solved again. The procedure terminates, when no more of these constraints can be identified. If the solution is still not integral, the solution space is divided by branching in the Branch & Bound procedure. The cut procedure should be performed in each node in the branching tree, before a new branching is performed.

In [5] cuts named *2-path cuts* are used as cuts in the Branch & Cut procedure.

First the general *k-path cut* is introduced in the following.

The set $\mathcal{S}$ is a subset of $\mathcal{V}$, such that $\mathcal{S} \subseteq \mathcal{V}$ and $|\mathcal{S}| \geq 1$. The flow into $\mathcal{S}$ is noted $X(\mathcal{S})$ and can be described as in Equation (9.7).

$$X(\mathcal{S}) = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{V} \setminus \mathcal{S}} x_{ij} \quad \forall \quad \mathcal{S} \subseteq \mathcal{V}, \mathcal{S} \neq \emptyset \tag{9.7}$$

The minimum number of pack machines needed to service the customers in the set $\mathcal{S}$ is noted $k(\mathcal{S})$. A constraint can be defined, by $X(\mathcal{S})$ and $k(\mathcal{S})$ as presented in Equation (9.8), since it is required, that at least $k(\mathcal{S})$ arcs enters $\mathcal{S}$ in a feasible integer solution.

$$X(\mathcal{S}) \geq k(\mathcal{S}) \tag{9.8}$$

It is difficult to apply the $k$-path cuts to the problem, since the number of sets $\mathcal{S}$ grows exponentially, according to the size of the solution space, and because it is $\mathcal{NP}$-hard to calculate $k(\mathcal{S})$, as stated in [5]. In [5] only sets where $k(\mathcal{S}) \geq 2$ and $X(\mathcal{S}) < 2$ are investigated, hence the name 2-path cut. When vehicles have limitated capacities according to the demand, which should be provided to the customers, it is easy to investigate whether $k(\mathcal{S}) > 1$. This is not the case in the pack machine problem. Instead it could be checked, whether a corresponding Traveling Salesman Problem with Time Windows is feasible in the set $\mathcal{S}$. This problem is $\mathcal{NP}$-hard, but could be solved by dynamical programming for small instances of $\mathcal{S}$ according to [5]. The flow $X(\mathcal{S})$ can be found by solving maximum flow problems between a customer $i$, where $i \in \mathcal{S}$ and a customer $j \in \mathcal{V} \setminus \mathcal{S}$.

### 9.2.3 Branching Strategies

If the solution is still fractional, branching is used on the fractional parts of the solution. Some of the possible branching strategies are described briefly in the next paragraphs. A further description of the branching strategies can be seen in article [18].

**Branching on the Number of Pack Machines.** If the number of pack machines used, noted as $m_{used} = v$, in a given solution is fractional, the solution is also fractional. The constraints $m_{used} \leq \lfloor v \rfloor$ and $m_{used} \geq \lceil v \rceil$ are added to the two new problems respectively. The constraints are added to the resulting subproblems in the Column Generation.

**Branching on Cost.** If all costs $c_{ij}$ are integers in the VRPTW, then the total cost of a feasible solution will be integer as well. If the cost of a solution $c_x = y$ is fractional, the constraints $c_x \leq \lfloor y \rfloor$ and $c_x \geq \lceil y \rceil$ should be added to the two new problems. These constraints are also added to the resulting subproblems in the Column Generation.

**Branching on Arcs.** If a variabel $x_{ij}$ is fractional, it could be fixed to either 0 or 1. If $x_{ij}$ is fixed at zero, it is removed from the network in the subproblem, and already generated routes including this arc, would be penalized, such that they still are feasible but expensive. If $x_{ij}$ is fixed to 1, the arcs $(i,k) \in \mathcal{A}$, $k \neq j$ and the arcs $(l,j) \in \mathcal{A}$, $l \neq i$ are removed in the same manner, as $(i,j)$ was, when fixed to 0.

Chapter 10

# Discussion

In this chapter an overview of the achievements in this thesis is presented in Section 10.1. Section 10.2 gives some considerations about the relevance of the project, and what kind of problems is possible to occur, when implementing the procedure. Some ideas for further work concerning model extensions and improvements of the program are presented in Section 10.3.

## 10.1 Contribution

The pack machine problem has been clarified and formulated with two approaches, that is the VRP and the VRPTW approach. The limited pack machine problem has been formulated with an TSPTW approach, which fulfilled the guidelines provided by Danisco. Solution methods have been developed for the pack machine problem, and tested on the pack machine and the limited pack machine problem.

The solution methods are based on heuristics and metaheuristics, since it is known, that these methods provide good solutions fast, which is required for the solution methods in this thesis. The implementation of the solution methods is probably improvable, as the computation time for the runs is high according to the complexity of the methods. The solution methods are described thoroughly, such that the implementation could be improved by Danisco, if lower computation time is of importance.

The results clarified, that the best results were obtained by Steepest Descent. It is possible, that Simulated Annealing could perform better if developed further.

Furthermore an exact method for the pack machine problem has been developed and described.

## 10.2 Considerations

Implementing the procedure might cause some troubles for Danisco, this is caused by conflicting objectives by the different interested parties in the project, this is described further in Section 10.2.1. The relevance of this project for Danisco is described in Section 10.2.2.

### 10.2.1 Conflicting Objectives

The pack machine problem was provided by Danisco, hence the objective of the problem is defined by Danisco. The pack machine problem has other interested parties than Danisco, that is for example those who operate the pack machines and the customers. The objective of the problem is different according to the point of view of the different stakeholders. E.g. the pack machine operators and the customers might have other objectives for the problem than Danisco.

When minimizing the total amount of time used, the pack machine operator is more effective, i.e. he has to visit more customers per day, and even though the distance travelled is smaller, this might give a worse solution seen from the pack machine operator's point of view.

The objective is also different seen from the customers point of view. The possibility of choosing time windows are preferable for the customers, and the narrower the time windows are the better, as narrow time windows means, that the customers can decide when to be serviced by the pack machines more precisely. This is important for the customers, because they are dependent of when the bulk is provided, according to the amount of feed needed for the animals.

### 10.2.2 Relevance of Project

The relevance of this project for Danisco is not as high as expected initially. A lot of effort was put in the VRPTW model, as this seemed to be the most adequate model. The limited pack machine problem was introduced very late in the progress of the project, such that only limited effort was used on this approach. This is a shame, as the limited pack machine problem, is the relevant problem for Danisco in the future.

Another approach could be to model the problem with precedence constraints. These could force the customers in the northern part of Sweden to be visited first, and the double visits to be separated, without discarding solutions which are feasible according to the pack machine problem, but infeasible in the VRPTW formulation. This formulation would be more general than the VRPTW formulation, as it could be formulated without using knowledge of the solution provided by Danisco.

## 10.3 Further Work

In this section some possible improvements of the project is presented. Two types of improvements are described, the first type of improvements is extensions of the model, these are presented in Section 10.3.1. The second type is improvements of the solution methods, these are presented in Section 10.3.2. Exact methods as presented in Chapter 9 are not considered here, since the focus are on improvements, which are of relevance for Danisco, instead of academic improvements.

### 10.3.1 Model Extensions

The model could be improved such, that the limitations described in Section 1.5, are included in the model.

The problem is implemented as a static problem. But the neighbourhood has a structure, which makes it easy to improve the problem, to contain dynamic orders as well as static orders. Implementing a forecast for customers which arrive or regret orders after the start of the campaign seems like a more difficult task. As the number of customers which arrives after the campaign is started is low, it is rather easy to insert these customers in the schedule manually, hence it does not seem that interesting to include dynamic features in the solution method.

Euclidean distances are used as a measure of the distances between the customers. If the actual road distances were used, the problem would be more realistic. The implementation, can handle euclidean distances as well as road distances, if these were available. such that only minor changes in the program are neccesary, if road distances are used. It is however a comprehensive task to collect the road distances.

Including the trucks in the problem, seems to give rise to a very different problem. Both the object function and the constraints should be reformulated in the mathematical model, as the objective is to minimize the total time used for both pack machines and trucks. Furthermore the bulk should be provided to the customers by the trucks, at the same time or immediately after the pack machine arrives. This should be formulated as constraints. As there are a limited number of trucks, it is not possible to send all pack machines to for example the northern part of Sweden at the same time, as the transportation time for the trucks would be larger, and more trucks are needed to provide bulk to the customers. The constraint could be formulated, such that the sum of the distances between the pack machines and the depot must not exceed a given value. The limited keeping quality also seems hard to incorporate in the model. Constraints guaranteeing that the bulk is provided to the customers and packed immediately should be added to the model.

The idea of including the Danish customers in the problem, seems even more important in the limited pack machine problem. This is due to the close down of the factories in Denmark and Sweden, which makes the problems in both countries more difficult to solve, as the routes should be scheduled better, when only two pack machines are available. Hence the advantage of solving the integrated problem of Danish and Swedish customers is greater.

Neither break downs of the factory nor of the pack machines are considered. If a break down occurs, it is always possible to solve the problem for the customers in the remaining part of the campaign. It does not seem reasonable to incorporate a forecast procedure for break downs, as these are rare according to Danisco.

Personal considerations are a difficult task when formulating a problem. It could be preferable, if the schedule are made such, that the pack machine operator is near a town at close down. This is important, as the operator wants to spend the night at an appropriate location.

Some interesting experiments could be done by varying the narrow time windows. It would be of relevance, if great optimization is gained, when the time windows are extended to more than +/- 5 days. A weighted cost function could be used to evaluate the improvement of the routes compared to the reduced satisfaction seen from the customers view.

### 10.3.2 Solution Methods

The results obtained with the implemented solution methods, are not of a quality as expected. As seen from the results presented in Section 8.2.3, especially the metaheuristic Simulated Annealing fails to find good solutions. The results are much worse than expected, but they might get better if the solution methods are improved. In the first paragraphs, different approaches to improve the implemented solution methods are suggested. In the last paragraph it is described how the metaheuristic Tabu Search could be applied to the problem, and how this metaheuristic should be able to obtain better solutions than Simulated Annealing, for the pack machine problem. Some of the improvements seems easy to implement in the program, but the lack of time made this impossible.

**Improving Simulated Annealing.** The implementation of Simulated Annealing might be improved by performing a more comprehensive parameter tuning. This is time consuming and therefore not carried out in this project. Simulated Annealing can furthermore be improved, by implementing the changes described in the following paragraphs.

**Improving the Initial Solution.** The initial solution is obtained by an nearest neighbour construction heuristic, and improved by the insertion heuristic. In the nearest neighbour construction heuristic, the customers are added to the routes sequentially, but as described in Section 5.2.3, this could give some rather bad solutions, where the quality of the last route is really poor. This could be avoided, by adding customers to the routes in a parallel manner. The performance of Simulated Annealing, could be improved, if the initial solution has a better quality. This is however not guaranteed, but implementing a parallel version of the construction heuristic still seems like a good idea, especially as this could be done without too great an effort.

**Combining Steepest Descent and Simulated Annealing.** In article [16] a heuristic based on the combination of the methods Steepest Descent and Simulated Annealing is described. As seen in Section 8.2.3, the results obtained by Simulated Annealing are worse, than those obtained by Steepest Descent.

By improving the solution as much as possible with Steepest Descent and then proceeding with Simulated Annealing, this is avoided. A combination of the two methods seems attractive, if more time were available.

**Tabu Search.** Instead of improving Simulated Annealing, other metaheuristics could be used to solve the problem. The metaheuristic *Tabu Search* has been used with great success to solve VRPs and VRPTWs [3]. The ideas behind Tabu Search is described briefly in the next paragraphs, a more satisfying description of Tabu Search is found in for example [23] or [4]. The metaheuristic Tabu Search could be described as an extended Steepest Descent method. Steepest Descent terminates, when a local optimum is reached. The idea in Tabu Search is to continue, when a local optimum is reached. This could be done, by choosing the best solution in the neighbourhood, even though this solution is worse than the solution in the local optimum. From the new worse solution, the best solution in the neighbourhood is found. The probability of choosing the same solution as before, i.e. the solution in the local optimum, is great. If this happens, the algorithm would switch between these two solutions, and no further improvements would be gained. To avoid returning to old solutions, representations of such solutions are defined as *Tabu*. A move is then prohibited in the next iterations after the inverse move was performed. The Tabus are stored in a *Tabu list* and the number of iterations where a move is prohibited, is decided by the *Tabu length*, which should be chosen by parameter tuning, as the parameters in Simulated Annealing. The Tabus are removed from the list with the *First In First Out* procedure. Sometimes other solutions than intended are discarded as Tabu. An *aspiration criterion* could be used to conquer this, such that for example a solution, which is better than the so far best found solution, could be chosen, even though it is Tabu.

If Tabu Search was used on the pack machine problem, it would be guaranteed, that the local optimum found by Steepest Descent, would be reached assuming at least the same number of iterations is used. As the method performs as Steepest Descent until, the first local optimum is reached.

Chapter 11

# Conclusion

In this thesis, motivation has been presented for solving the pack machine problem and the limited pack machine problem with methods from operational research. The motivation for solving the limited pack machine problem with operational research is even greater, as no solution exists, which could be used as a template for the problem. The pack machine problem has been described, and the limited pack machine problem has been presented.

Two different models for formulating the pack machine problem have been presented. The models are based on the VRP and the VRPTW model respectively. The model based on the VRP approach has the disadvantage of not separating the double visits. The VRPTW formulation separates the double visits and forces the customers in the northern part of Sweden to be visited first. Furthermore the wide time windows are adhered. It is a problem, that the narrow time windows were obtained from a feasible solution provided by Danisco, therefore this model can not be applied directly on the limited pack machine problem, as no feasible solution was provided by Danisco. The approach for formulating the limited pack machine problem were provided by Danisco. As the problem only were known late in the project, there has been no time for developing better models.

The complexity of the different models have been described, and it has been detected that the formulations of the problems are among the most difficult to solve. The models have been compared and the conclusion is, that the VRPTW approach is best for modelling the pack machine problem, but the VRPTW approach is not possible when modelling the limited pack machine problem, as no solution has been provided for this problem.

A litterature review has been presented, as there are no litterature on the pack machine and the limited pack machine problem, the review concerns litterature on the approaches for modelling the problems. The amount of litterature on the VRP, VRPTW and TSP is extensive, though only few articles describe how the problems could be solved with Simulated Annealing.

Several solution methods have been proposed for solving the pack machine problem. For the VRP approach it seems difficult to apply the saving heuristic, while the two-phase heuristic seems more adequate for the problem, as the effectivity of the pack machines could be taken into account when dividing the customers initially. It has been presented how the routes could be scheduled by using the nearest neighbour heuristic, and a disadvantage of using this heuristic has also been presented. Furthermore it has been presented how the routes could be improved by the 2-opt procedure.

A lot of work have been put into the solution methods of the VRPTW formulation. Two different solution methods, that is Simulated Annealing and Steepest Descent have been described thoroughly. It has been described how an initial solution could be obtained by a sequential construction heuristic. A problem connected with the sequential construction of the routes has been described. It has also been described how the solution could be improved with an insertion heuristic in case of infeasibility. The insertion heuristic uses a push forward calculation and a feasibility check, which has also been described in this report. Several neighbourhoods have been presented for the heuristics, and the relocate operator has been chosen as neighbourhood, as this had the ability of handling infeasibility, if not all customers were scheduled in the initial solution. Furthermore this neighbourhood could be used in a dynamic version of the problem. A push backward operation has been derived to the neighbourhood. Furthermore different evaluation measures of the solution have been presented, it has been discussed how the infeasibility of a solution could be measured. Finally different stopping criteria have been presented for Simulated Annealing. It has been chosen to terminate Simulated Annealing after a specified number of iterations, as this would give a rightful comparison, when using different values of the parameters.

The limited pack machine problem were known too late to develop new solution methods. Hence it has been evaluated how solution methods for the pack machine problem could be used to solve this problem.

In the parameter tuning test data sets developed from the original data have been used. It has been considered whether this was the right approach, but it has been chosen as a golden mean. From the parameter tuning of the nearest neighbour construction heuristic it has been seen, that some of the best solutions were obtained by only using the geographical distance as a measure instead of the weighted cost function. The fact that the time windows have been obtained from a solution supposed to be good, seem to explain this. It has been more difficult to see clear tendencies for Simulated Annealing, though a low intial temperature seemed like a good choice.

The results for the VRP approach has been as expected. It has been seen that the 2-opt method could improve the solution considerably. Furthermore the problem concerning double visits in the VRP formulation has been observed. For the VRPTW formulation it has been seen, that the solution obtained by nearest neighbour was infeasible, as not all customers were scheduled. After improving the solution with the insertion heuristic, the solution has become feasible. The obtained solution has been worse than the solution provided by Danisco according to the total distance, but better according to the amount of time used, which was not expected. This is caused by the fact, that the sequential heuristic assign fewer customers to the route, which is driven by the least effective pack machine.

When using Steepest Descent to minimize the amount of time used in the solution, it has been seen that the unlimited neighbourhood performs slightly better than the limited neighbourhood. When using Steepest Descent to minimize the distance, it has been seen that the methods terminates in a local optimum which is worse than the solution provided by Danisco when improving the obtained solution. Furthermore it has been seen, that the solution provided by Danisco could be improved considerably by using the unlimited neighbourhood. The results obtained with Simulated Annealing were much worse than expected. Only limited improvements have been seen, when using the parameters chosen from the parametertuning, even though the number of iterations were raised to 50000. Tuning the initial temperature after the acceptance ratio, indicated that the the initial temperature was to low. But raising the temperature to a more appropriate level according to the acceptance ratio, did not seem to improve the performance. The conclusion must be, that further work is needed on Simulated Annealing, if better results should be obtained.

It have been difficult to evaluate the results for the limited pack machine problem, as it seems like the interpretation of data was wrong. It has been seen, that a feasible solution was obtained for the customers in the rest of Sweden. The solution for the customers in Scania was not feasible, as all customers should be serviced in the first part of the campaign. The solutions were improved slightly by Steepest Descent, but it has been seen, that the resulting routes are still improvable.

Appendix A

# Data

## A.1 Map of Sweden


Figure A.1: A map of Sweden provided by Danisco, where the different geographic zones are seen.

## A.2 The Data Set Provided By Danisco

| ID | X-coord | Y-coord | Machine nu | Order no. | G.zo | Dlv.date | Order qty |
|---|---|---|---|---|---|---|---|
| 1050234 | 618715 | 136790 | 1 | 480506 | 401 | 01.08.2005 | 105.000 |
| 1181123 | 615800 | 140690 | 2 | 481785 | 401 | 01.09.2005 | 180.001 |
| 1047192 | 617500 | 137600 | 2 | 481554 | 401 | 01.09.2005 | 65.001 |
| 1081472 | 617830 | 133290 | 1 | 480530 | 401 | 01.09.2005 | 70.000 |
| 1064211 | 615150 | 135295 | 1 | 480985 | 401 | 01.09.2005 | 70.000 |
| 1160267 | 616160 | 138020 | 2 | 481700 | 401 | 01.09.2005 | 70.001 |
| 1051793 | 618935 | 136360 | 1 | 480514 | 401 | 01.09.2005 | 110.000 |
| 1103584 | 614840 | 139880 | 2 | 481168 | 401 | 01.09.2005 | 130.001 |
| 1018622 | 619800 | 134200 | 1 | 481588 | 401 | 01.09.2005 | 105.000 |
| 1046082 | 616120 | 137440 | 2 | 481002 | 401 | 01.09.2005 | 70.001 |
| 1320117 | 621535 | 142815 | 2 | 481241 | 401 | 01.09.2005 | 105.000 |
| 1051955 | 618905 | 136100 | 1 | 481991 | 401 | 01.09.2005 | 100.000 |
| 1110728 | 619130 | 139027 | 2 | 482116 | 401 | 01.09.2005 | 65.001 |
| 1052706 | 618390 | 135540 | 1 | 481930 | 401 | 01.09.2005 | 140.001 |
| 1206443 | 627010 | 133780 | 1 | 480869 | 402 | 01.09.2005 | 70.000 |
| 1105534 | 615310 | 138260 | 2 | 482027 | 401 | 01.09.2005 | 200.000 |
| 1008724 | 620720 | 132575 | 1 | 481733 | 401 | 01.09.2005 | 65.000 |
| 1044200 | 615160 | 136840 | 2 | 480524 | 401 | 01.09.2005 | 100.000 |
| 1540173 | 626630 | 153890 | 3 | 480940 | 402 | 01.09.2005 | 105.000 |
| 1540173 | 626630 | 153890 | 3 | 496063 | 402 | 01.09.2005 | 105.000 |
| 1052476 | 618150 | 135380 | 1 | 483115 | 401 | 01.09.2005 | 65.000 |
| 1108423 | 616820 | 140070 | 2 | 481037 | 401 | 01.09.2005 | 175.000 |
| 1066205 | 619195 | 136880 | 1 | 480724 | 401 | 01.09.2005 | 105.000 |
| 1160315 | 616250 | 138540 | 2 | 480591 | 401 | 01.09.2005 | 70.000 |
| 1138563 | 623860 | 132315 | 1 | 481589 | 401 | 01.09.2005 | 65.000 |
| 1160598 | 616150 | 138060 | 2 | 480613 | 401 | 01.09.2005 | 140.000 |
| 1110916 | 619477 | 139654 | 2 | 480484 | 401 | 01.09.2005 | 105.000 |
| 1105616 | 615410 | 138090 | 2 | 480607 | 401 | 01.09.2005 | 70.001 |
| 1115330 | 621150 | 141030 | 2 | 486032 | 401 | 01.09.2005 | 250.000 |
| 1116370 | 621790 | 139955 | 2 | 486081 | 401 | 01.09.2005 | 315.000 |
| 1144082 | 624910 | 131520 | 1 | 481797 | 401 | 01.09.2005 | 100.000 |
| 1045704 | 615480 | 137850 | 2 | 481197 | 401 | 01.09.2005 | 140.000 |
| 1064187 | 615470 | 135470 | 1 | 481803 | 401 | 01.09.2005 | 105.000 |
| 1105400 | 615270 | 138880 | 2 | 481106 | 401 | 01.09.2005 | 245.001 |
| 1122204 | 621540 | 139095 | 2 | 481810 | 401 | 01.09.2005 | 105.000 |
| 1131593 | 623160 | 138495 | 2 | 481172 | 401 | 01.09.2005 | 100.000 |
| 1107082 | 616236 | 139464 | 2 | 481670 | 401 | 01.09.2005 | 140.000 |
| 1052527 | 618190 | 135890 | 1 | 481092 | 401 | 01.09.2005 | 70.001 |
| 1008664 | 620627 | 132677 | 1 | 481901 | 401 | 01.09.2005 | 105.000 |
| 1104273 | 614180 | 139480 | 2 | 480577 | 401 | 01.09.2005 | 100.000 |
| 1036728 | 615555 | 133155 | 1 | 481791 | 401 | 01.09.2005 | 130.001 |
| 1016724 | 619980 | 133360 | 1 | 480886 | 401 | 01.09.2005 | 105.000 |
| 1115055 | 620765 | 140890 | 2 | 486033 | 401 | 01.09.2005 | 105.000 |
| 1105605 | 615420 | 138160 | 2 | 481660 | 401 | 01.09.2005 | 300.000 |
| 1048711 | 617656 | 136942 | 2 | 481520 | 401 | 01.09.2005 | 250.000 |
| 1530241 | 626640 | 151980 | 3 | 481533 | 402 | 01.09.2005 | 105.001 |
| 1203177 | 625881 | 132773 | 1 | 481736 | 402 | 01.09.2005 | 140.001 |
| 1107527 | 615760 | 138950 | 2 | 481434 | 401 | 01.09.2005 | 105.001 |
| 1108963 | 617510 | 139730 | 2 | 480969 | 401 | 01.09.2005 | 210.001 |
| 1116166 | 621470 | 139985 | 2 | 480535 | 401 | 01.09.2005 | 140.000 |
| 1116640 | 622140 | 140625 | 2 | 487357 | 401 | 01.09.2005 | 220.000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1045132 | 615100 | 137390 | 2 | 483108 | 401 | 01.09.2005 | 140.000 |
| 1115400 | 621980 | 141390 | 2 | 481102 | 401 | 01.09.2005 | 175.000 |
| 1107187 | 616050 | 139280 | 2 | 481080 | 401 | 01.09.2005 | 245.000 |
| 1145368 | 625582 | 130591 | 1 | 481042 | 401 | 01.09.2005 | 135.000 |
| 1203637 | 625970 | 132720 | 1 | 481047 | 402 | 01.09.2005 | 105.000 |
| 1132375 | 621830 | 138307 | 2 | 481104 | 401 | 01.09.2005 | 100.000 |
| 1203176 | 625556 | 133171 | 1 | 481045 | 402 | 01.09.2005 | 105.000 |
| 1109197 | 617170 | 138770 | 2 | 480958 | 401 | 01.09.2005 | 450.000 |
| 1066185 | 619178 | 136807 | 1 | 480979 | 401 | 01.09.2005 | 105.000 |
| 1014516 | 620100 | 133020 | 1 | 481932 | 401 | 01.09.2005 | 105.000 |
| 1166041 | 622772 | 133192 | 1 | 480892 | 401 | 01.09.2005 | 260.000 |
| 1132000 | 622960 | 138173 | 2 | 483124 | 401 | 01.09.2005 | 70.000 |
| 1047287 | 616821 | 137102 | 2 | 483119 | 401 | 01.09.2005 | 70.000 |
| 1107046 | 615440 | 139740 | 2 | 482125 | 401 | 01.09.2005 | 210.001 |
| 1050234 | 618715 | 136790 | 1 | 480512 | 401 | 01.09.2005 | 105.000 |
| 1107117 | 616280 | 139320 | 2 | 481133 | 401 | 01.09.2005 | 70.000 |
| 1004158 | 622950 | 131080 | 1 | 481726 | 401 | 01.09.2005 | 175.000 |
| 1047869 | 616939 | 137885 | 2 | 481603 | 401 | 01.09.2005 | 70.000 |
| 1101610 | 616303 | 139974 | 2 | 483125 | 401 | 01.09.2005 | 200.000 |
| 1140403 | 622490 | 132885 | 1 | 481198 | 401 | 01.09.2005 | 175.000 |
| 1203502 | 626050 | 132680 | 1 | 480713 | 402 | 01.09.2005 | 70.000 |
| 1052392 | 617880 | 135965 | 1 | 482023 | 401 | 01.09.2005 | 105.000 |
| 1181649 | 616030 | 140430 | 2 | 481708 | 401 | 01.09.2005 | 70.000 |
| 1047504 | 616670 | 137680 | 2 | 484331 | 401 | 01.09.2005 | 315.000 |
| 430341 | 625539 | 131197 | 1 | 481728 | 401 | 01.09.2005 | 70.000 |
| 420008 | 624683 | 131432 | 1 | 481906 | 401 | 01.09.2005 | 175.000 |
| 421000 | 627150 | 152330 | 3 | 481800 | 402 | 01.09.2005 | 140.000 |
| 428002 | 616050 | 137930 | 2 | 481605 | 401 | 01.09.2005 | 140.001 |
| 428004 | 616240 | 139010 | 2 | 480593 | 401 | 01.09.2005 | 140.001 |
| 428009 | 618245 | 139062 | 2 | 481062 | 401 | 01.09.2005 | 245.000 |
| 429005 | 619785 | 134330 | 1 | 481486 | 401 | 01.09.2005 | 100.000 |
| 430113 | 621584 | 132860 | 1 | 482348 | 401 | 01.09.2005 | 100.000 |
| 430200 | 624147 | 133446 | 1 | 480934 | 401 | 01.09.2005 | 140.000 |
| 430217 | 622730 | 135124 | 1 | 481982 | 401 | 01.09.2005 | 70.001 |
| 430224 | 623530 | 134925 | 1 | 481513 | 401 | 01.09.2005 | 210.000 |
| 430239 | 623023 | 135167 | 1 | 481235 | 401 | 01.09.2005 | 105.000 |
| 430250 | 622573 | 131813 | 1 | 482007 | 401 | 01.09.2005 | 140.000 |
| 430251 | 623519 | 134792 | 1 | 481191 | 401 | 01.09.2005 | 280.000 |
| 430252 | 622187 | 133716 | 1 | 480968 | 401 | 01.09.2005 | 65.000 |
| 430305 | 625930 | 130870 | 1 | 481690 | 401 | 01.09.2005 | 130.000 |
| 430312 | 624750 | 132365 | 1 | 480810 | 401 | 01.09.2005 | 200.001 |
| 430333 | 624597 | 131337 | 1 | 481387 | 401 | 01.09.2005 | 70.000 |
| 430340 | 625568 | 131167 | 1 | 481783 | 401 | 01.09.2005 | 70.000 |
| 430404 | 626610 | 132980 | 1 | 480561 | 402 | 01.09.2005 | 70.000 |
| 430406 | 626315 | 133433 | 1 | 480753 | 402 | 01.09.2005 | 70.000 |
| 430422 | 626937 | 135313 | 1 | 480857 | 402 | 01.09.2005 | 105.000 |
| 430431 | 627668 | 133723 | 1 | 482062 | 402 | 01.09.2005 | 140.000 |
| 430438 | 626827 | 133600 | 1 | 481136 | 402 | 01.09.2005 | 70.000 |
| 430446 | 626277 | 134013 | 1 | 481451 | 402 | 01.09.2005 | 70.000 |
| 430451 | 627850 | 132768 | 1 | 481367 | 402 | 01.09.2005 | 105.000 |
| 430459 | 625485 | 133152 | 1 | 483271 | 402 | 01.09.2005 | 65.000 |
| 430521 | 628607 | 132931 | 1 | 480647 | 402 | 01.09.2005 | 165.001 |
| 430525 | 628805 | 131316 | 1 | 481071 | 402 | 01.09.2005 | 175.000 |
| 430541 | 630051 | 131474 | 1 | 480619 | 402 | 01.09.2005 | 100.000 |
| 430542 | 630298 | 131424 | 1 | 481703 | 402 | 01.09.2005 | 70.000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 430545 | 632278 | 130446 | 1 | 481187 | 402 | 01.09.2005 | 70.000 |
| 430563 | 630533 | 130668 | 1 | 481798 | 402 | 01.09.2005 | 140.000 |
| 430574 | 634039 | 131120 | 1 | 481706 | 402 | 01.09.2005 | 70.000 |
| 430579 | 628473 | 132639 | 1 | 481636 | 402 | 01.09.2005 | 35.000 |
| 430583 | 633180 | 129159 | 1 | 480887 | 402 | 01.09.2005 | 175.000 |
| 430587 | 632041 | 129877 | 1 | 481807 | 402 | 01.09.2005 | 350.000 |
| 430590 | 629300 | 134451 | 1 | 480590 | 402 | 01.09.2005 | 280.000 |
| 430612 | 631338 | 137385 | 1 | 481424 | 402 | 01.09.2005 | 210.000 |
| 430621 | 630588 | 136565 | 1 | 481354 | 402 | 01.09.2005 | 210.000 |
| 430622 | 641421 | 139530 | 1 | 481429 | 403 | 01.09.2005 | 250.000 |
| 430623 | 635141 | 139375 | 1 | 481032 | 402 | 01.09.2005 | 65.000 |
| 430625 | 633961 | 136753 | 1 | 481338 | 402 | 01.09.2005 | 140.000 |
| 430628 | 633509 | 135417 | 1 | 480960 | 402 | 01.09.2005 | 140.000 |
| 430704 | 645899 | 132342 | 1 | 481249 | 404 | 01.09.2005 | 665.000 |
| 430708 | 639166 | 130045 | 1 | 481638 | 403 | 01.09.2005 | 250.000 |
| 430815 | 645526 | 139303 | 1 | 481161 | 404 | 01.09.2005 | 70.000 |
| 430816 | 645765 | 139354 | 1 | 481910 | 404 | 01.09.2005 | 70.000 |
| 430824 | 646035 | 139497 | 1 | 482378 | 404 | 01.09.2005 | 140.000 |
| 430829 | 645429 | 139817 | 1 | 481258 | 404 | 01.09.2005 | 175.000 |
| 430830 | 645877 | 138956 | 1 | 481819 | 404 | 01.09.2005 | 70.000 |
| 440617 | 615931 | 135689 | 1 | 484332 | 401 | 01.09.2005 | 70.001 |
| 460202 | 618050 | 137640 | 2 | 481095 | 401 | 01.09.2005 | 140.000 |
| 460207 | 617980 | 137800 | 2 | 481237 | 401 | 01.09.2005 | 100.000 |
| 460221 | 617290 | 137880 | 2 | 480750 | 401 | 01.09.2005 | 130.000 |
| 460224 | 616960 | 137810 | 2 | 481601 | 401 | 01.09.2005 | 105.000 |
| 460228 | 617002 | 137505 | 2 | 482004 | 401 | 01.09.2005 | 175.001 |
| 460242 | 616620 | 137380 | 2 | 481087 | 401 | 01.09.2005 | 210.000 |
| 460263 | 617230 | 137210 | 2 | 480865 | 401 | 01.09.2005 | 110.000 |
| 460290 | 617400 | 138300 | 2 | 481101 | 401 | 01.09.2005 | 150.000 |
| 460291 | 617200 | 137600 | 2 | 481759 | 401 | 01.09.2005 | 175.001 |
| 460341 | 617531 | 136785 | 2 | 480789 | 401 | 01.09.2005 | 175.000 |
| 460401 | 617454 | 138797 | 2 | 481595 | 401 | 01.09.2005 | 210.001 |
| 460405 | 616880 | 140280 | 2 | 481137 | 401 | 01.09.2005 | 210.000 |
| 460413 | 617454 | 138827 | 2 | 481085 | 401 | 01.09.2005 | 140.000 |
| 460419 | 617300 | 138440 | 2 | 481549 | 401 | 01.09.2005 | 140.000 |
| 460423 | 617350 | 138620 | 2 | 480826 | 401 | 01.09.2005 | 70.000 |
| 460429 | 618948 | 138458 | 2 | 480893 | 401 | 01.09.2005 | 70.000 |
| 460601 | 615600 | 136540 | 2 | 481926 | 401 | 01.09.2005 | 100.000 |
| 460814 | 617076 | 139275 | 2 | 480662 | 401 | 01.09.2005 | 65.000 |
| 460818 | 616500 | 139180 | 2 | 481132 | 401 | 01.09.2005 | 65.001 |
| 470101 | 622230 | 150060 | 3 | 481455 | 402 | 01.09.2005 | 175.001 |
| 470206 | 627660 | 151530 | 3 | 480783 | 402 | 01.09.2005 | 145.001 |
| 470423 | 627340 | 155320 | 3 | 481077 | 402 | 01.09.2005 | 70.000 |
| 470430 | 630204 | 156101 | 3 | 480878 | 402 | 01.09.2005 | 140.000 |
| 470506 | 630095 | 155274 | 3 | 481031 | 402 | 01.09.2005 | 140.002 |
| 470906 | 632020 | 156300 | 3 | 481512 | 402 | 01.09.2005 | 140.000 |
| 470521 | 628346 | 154493 | 3 | 481181 | 402 | 01.09.2005 | 175.000 |
| 470540 | 627077 | 151825 | 3 | 482073 | 402 | 01.09.2005 | 140.001 |
| 470544 | 626489 | 154864 | 3 | 480943 | 402 | 01.09.2005 | 150.000 |
| 470552 | 631116 | 153458 | 3 | 480551 | 402 | 01.09.2005 | 72.000 |
| 470557 | 627756 | 155316 | 3 | 481205 | 402 | 01.09.2005 | 175.000 |
| 470572 | 640969 | 152242 | 3 | 481264 | 404 | 01.09.2005 | 175.000 |
| 470584 | 641620 | 140958 | 1 | 481371 | 403 | 01.09.2005 | 75.000 |
| 470586 | 641647 | 152142 | 3 | 482056 | 403 | 01.09.2005 | 460.000 |
| 470588 | 628617 | 154250 | 3 | 480873 | 402 | 01.09.2005 | 245.000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 470596 | 633165 | 145340 | 2 | 480540 | 402 | 01.09.2005 | 70.001 |
| 470596 | 633165 | 145340 | 2 | 481634 | 402 | 01.09.2005 | 70.000 |
| 470597 | 630924 | 142098 | 2 | 481385 | 402 | 01.09.2005 | 70.000 |
| 470598 | 636565 | 143817 | 1 | 481256 | 403 | 01.09.2005 | 106.000 |
| 470599 | 628351 | 145502 | 2 | 481871 | 402 | 01.09.2005 | 315.001 |
| 470600 | 639765 | 150769 | 1 | 481346 | 403 | 01.09.2005 | 70.000 |
| 470602 | 648640 | 150601 | 3 | 481039 | 404 | 01.09.2005 | 70.001 |
| 470622 | 627830 | 155300 | 3 | 480708 | 402 | 01.09.2005 | 70.001 |
| 470623 | 639189 | 142649 | 1 | 480657 | 403 | 01.09.2005 | 140.000 |
| 470629 | 642434 | 145936 | 3 | 489207 | 404 | 01.09.2005 | 105.000 |
| 470633 | 642338 | 154133 | 3 | 481597 | 404 | 01.09.2005 | 140.000 |
| 470635 | 629928 | 155230 | 3 | 481904 | 402 | 01.09.2005 | 150.000 |
| 470637 | 629138 | 152730 | 3 | 480765 | 402 | 01.09.2005 | 70.000 |
| 470646 | 636351 | 154799 | 3 | 481993 | 403 | 01.09.2005 | 70.001 |
| 470650 | 631009 | 155649 | 3 | 482351 | 402 | 01.09.2005 | 70.000 |
| 470658 | 635086 | 145275 | 1 | 482060 | 403 | 01.09.2005 | 230.000 |
| 470661 | 630209 | 141725 | 2 | 482304 | 402 | 01.09.2005 | 210.001 |
| 470664 | 628802 | 154591 | 3 | 481079 | 402 | 01.09.2005 | 105.000 |
| 470671 | 626586 | 152159 | 3 | 480623 | 402 | 01.09.2005 | 245.001 |
| 470215 | 648700 | 153711 | 3 | 481914 | 404 | 01.09.2005 | 700.000 |
| 470703 | 626832 | 151970 | 3 | 481213 | 402 | 01.09.2005 | 140.002 |
| 470711 | 631111 | 152137 | 3 | 480787 | 402 | 01.09.2005 | 70.001 |
| 470717 | 646507 | 148113 | 3 | 481029 | 404 | 01.09.2005 | 110.001 |
| 470718 | 654292 | 150652 | 3 | 482282 | 405 | 01.09.2005 | 175.001 |
| 470719 | 631400 | 142133 | 2 | 481525 | 402 | 01.09.2005 | 70.001 |
| 470724 | 630665 | 139248 | 1 | 480805 | 402 | 01.09.2005 | 425.001 |
| 470735 | 640961 | 153217 | 3 | 480814 | 404 | 01.09.2005 | 175.000 |
| 470758 | 628959 | 152700 | 3 | 482051 | 402 | 01.09.2005 | 315.000 |
| 470759 | 635938 | 149822 | 1 | 481687 | 403 | 01.09.2005 | 175.000 |
| 470763 | 630569 | 139967 | 1 | 480815 | 402 | 01.09.2005 | 200.001 |
| 470782 | 634001 | 151097 | 3 | 482377 | 403 | 01.09.2005 | 105.000 |
| 470786 | 628461 | 154582 | 3 | 481243 | 402 | 01.09.2005 | 210.000 |
| 470804 | 639627 | 145189 | 1 | 487471 | 403 | 01.09.2005 | 350.000 |
| 490411 | 618800 | 137358 | 1 | 481388 | 401 | 01.09.2005 | 70.000 |
| 490413 | 618273 | 135789 | 1 | 481099 | 401 | 01.09.2005 | 105.000 |
| 490421 | 620982 | 136878 | 2 | 482002 | 401 | 01.09.2005 | 100.001 |
| 490422 | 619525 | 136855 | 1 | 481720 | 401 | 01.09.2005 | 100.001 |
| 490423 | 619257 | 137166 | 1 | 480745 | 401 | 01.09.2005 | 105.000 |
| 490436 | 621426 | 137079 | 2 | 481046 | 401 | 01.09.2005 | 140.000 |
| 490437 | 618580 | 136880 | 1 | 481201 | 401 | 01.09.2005 | 70.000 |
| 490450 | 620564 | 133877 | 1 | 481463 | 401 | 01.09.2005 | 70.000 |
| 490460 | 618290 | 138136 | 2 | 480990 | 401 | 01.09.2005 | 150.001 |
| 490471 | 618532 | 135883 | 1 | 480754 | 401 | 01.09.2005 | 70.000 |
| 490488 | 621587 | 137735 | 2 | 481188 | 401 | 01.09.2005 | 280.001 |
| 490496 | 618553 | 135330 | 1 | 481130 | 401 | 01.09.2005 | 70.001 |
| 490499 | 619435 | 136934 | 1 | 481368 | 401 | 01.09.2005 | 70.000 |
| 490501 | 618534 | 136326 | 1 | 481918 | 401 | 01.09.2005 | 65.000 |
| 491004 | 624430 | 137800 | 2 | 481096 | 401 | 01.09.2005 | 105.000 |
| 491009 | 622837 | 137860 | 2 | 480527 | 401 | 01.09.2005 | 70.000 |
| 491014 | 622107 | 139356 | 2 | 481091 | 401 | 01.09.2005 | 100.000 |
| 491026 | 622932 | 139200 | 2 | 481517 | 401 | 01.09.2005 | 100.000 |
| 491031 | 625256 | 138165 | 2 | 480705 | 401 | 01.09.2005 | 165.000 |
| 491037 | 623567 | 139354 | 2 | 481531 | 401 | 01.09.2005 | 175.001 |
| 491041 | 622718 | 137759 | 2 | 480634 | 401 | 01.09.2005 | 175.000 |
| 491060 | 623399 | 139452 | 2 | 480959 | 401 | 01.09.2005 | 100.000 |

```
491066 625442 138712 2 481761 401 01.09.2005 105.000
491068 623077 141913 2 480987 401 01.09.2005 70.000
491070 623644 138037 2 481689 401 01.09.2005 105.000
491076 623193 138340 2 480951 401 01.09.2005 100.000
491081 620687 137779 2 480885 401 01.09.2005 315.001
491083 624200 137761 2 480824 401 01.09.2005 100.000
491094 620385 139266 2 481511 401 01.09.2005 141.001
491096 622770 137804 2 481490 401 01.09.2005 150.000
491099 626097 141015 2 480702 401 01.09.2005 105.000
491101 626485 140695 2 482382 401 01.09.2005 105.000
496209 629881 131602 1 481409 402 01.09.2005 210.000
430597 633093 128778 1 480948 402 01.09.2005 140.000
497010 638414 130469 1 481865 403 01.09.2005 70.001
430466 625830 134073 1 480883 402 01.09.2005 70.000
470807 639111 145905 1 482068 404 01.09.2005 70.000
470813 636143 147217 1 482070 403 01.09.2005 105.000
481729 01.09.2005 140.000
470830 628519 142392 2 481056 402 01.09.2005 210.001
470845 637356 152355 3 481526 403 01.09.2005 70.000
470846 629045 152632 3 480531 402 01.09.2005 210.000
470847 624816 151541 3 480631 402 01.09.2005 70.001
470848 643667 152861 3 482021 404 01.09.2005 315.000
470856 639476 148718 1 481905 403 01.09.2005 105.000
470864 629210 155672 3 481394 402 01.09.2005 245.000
470859 628037 154460 3 481731 402 01.09.2005 142.000
470869 627420 154940 3 481180 402 01.09.2005 105.001
470869 627420 154940 3 481992 402 01.09.2005 105.000
470908 638076 154091 3 483090 403 01.09.2005 175.000
490448 621467 137754 2 481189 401 01.09.2005 100.000
430695 632144 129810 1 480629 402 01.09.2005 140.000
460238 617090 138430 2 481183 401 01.09.2005 65.000
470921 640437 149539 1 482356 403 01.09.2005 280.000
460436 618759 138982 2 481355 401 01.09.2005 130.000
470929 628810 151100 3 481250 402 01.09.2005 105.000
470932 626225 154725 3 481427 402 01.09.2005 105.000
470933 626796 151919 3 481410 402 01.09.2005 70.000
470936 630387 155285 3 481541 402 01.09.2005 105.001
491103 623011 138475 2 481093 401 01.09.2005 100.000
470942 643765 154712 3 481699 404 01.09.2005 105.001
440623 614834 131311 1 481498 401 01.09.2005 175.000
430505 633612 131048 1 480788 402 01.09.2005 105.001
470949 639703 140993 1 482064 403 01.09.2005 70.000
470950 626662 154923 3 480813 402 01.09.2005 70.001
430506 627923 133067 1 481529 402 01.09.2005 175.001
470958 628213 145113 2 481392 402 01.09.2005 105.000
430509 628692 132669 1 481050 402 01.09.2005 105.000
470960 630567 156212 3 481909 402 01.09.2005 180.000
470966 624756 153935 3 481864 402 01.09.2005 70.001
470969 624824 154522 3 481665 402 01.09.2005 105.001
496255 633726 132585 1 481812 402 01.09.2005 175.000
430649 633610 129379 1 480862 402 01.09.2005 105.000
470020 632260 142721 2 482353 402 01.09.2005 210.001
430904 650502 141835 1 481972 404 01.09.2005 560.000
430648 633114 128786 1 482297 402 01.09.2005 525.000
490512 618374 136256 1 481642 401 01.09.2005 130.000
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 470976 | 626933 | 152334 | 3 | 481384 | 402 | 01.09.2005 | 140.000 |
| 470976 | 626933 | 152334 | 3 | 494681 | 402 | 01.09.2005 | 105.000 |
| 470976 | 626933 | 152334 | 3 | 495052 | 402 | 01.09.2005 | 35.000 |
| 491105 | 621830 | 138410 | 2 | 480992 | 401 | 01.09.2005 | 65.000 |
| 460353 | 617241 | 138668 | 2 | 481193 | 401 | 01.09.2005 | 105.001 |
| 470984 | 634023 | 157558 | 3 | 481267 | 402 | 01.09.2005 | 105.000 |
| 470987 | 631538 | 144443 | 2 | 480937 | 402 | 01.09.2005 | 525.010 |
| 430470 | 633871 | 129179 | 1 | 480596 | 402 | 01.09.2005 | 300.000 |
| 470992 | 634457 | 157586 | 3 | 480889 | 402 | 01.09.2005 | 70.000 |
| 470996 | 630985 | 156410 | 3 | 481432 | 402 | 01.09.2005 | 140.000 |
| 470997 | 624175 | 147735 | 3 | 481818 | 402 | 01.09.2005 | 315.001 |
| 470998 | 632197 | 142709 | 2 | 482074 | 402 | 01.09.2005 | 140.001 |
| 470001 | 641649 | 144556 | 3 | 481389 | 404 | 01.09.2005 | 65.001 |
| 470012 | 631442 | 141726 | 2 | 481532 | 402 | 01.09.2005 | 210.000 |
| 470012 | 631442 | 141726 | 2 | 493416 | 402 | 01.09.2005 | 210.001 |
| 470010 | 650425 | 156860 | 3 | 483101 | 404 | 01.09.2005 | 105.000 |
| 470010 | 650425 | 156860 | 3 | 483273 | 404 | 01.09.2005 | 105.000 |
| 470011 | 645103 | 151085 | 3 | 481983 | 404 | 01.09.2005 | 175.001 |
| 470015 | 646488 | 147952 | 3 | 483270 | 404 | 01.09.2005 | 350.000 |
| 470015 | 646488 | 147952 | 3 | 506633 | 404 | 01.09.2005 | 175.000 |
| 430639 | 643412 | 133865 | 1 | 482295 | 404 | 01.09.2005 | 490.000 |
| 470051 | 635654 | 147007 | 1 | 481934 | 403 | 01.09.2005 | 140.000 |
| 470059 | 641734 | 150340 | 3 | 481980 | 404 | 01.09.2005 | 1.250.000 |
| 460361 | 616065 | 138118 | 2 | 480749 | 401 | 01.09.2005 | 140.000 |
| 430643 | 638520 | 133838 | 1 | 480823 | 403 | 01.09.2005 | 210.000 |
| 496258 | 630114 | 130865 | 1 | 480557 | 402 | 01.09.2005 | 210.000 |
| 496261 | 632515 | 140634 | 1 | 482044 | 402 | 01.09.2005 | 100.000 |
| 460367 | 622187 | 141888 | 2 | 481806 | 401 | 01.09.2005 | 140.000 |
| 460366 | 614582 | 139000 | 2 | 481564 | 401 | 01.09.2005 | 140.000 |
| 430850 | 643158 | 136514 | 1 | 481811 | 404 | 01.09.2005 | 295.000 |
| 430412 | 628862 | 132699 | 1 | 480651 | 402 | 01.09.2005 | 70.000 |
| 430644 | 625896 | 133251 | 1 | 482364 | 402 | 01.09.2005 | 105.000 |
| 470076 | 629336 | 154830 | 3 | 481683 | 402 | 01.09.2005 | 70.001 |
| 470079 | 622686 | 150504 | 3 | 480856 | 402 | 01.09.2005 | 315.001 |
| 470080 | 629804 | 153410 | 3 | 480774 | 402 | 01.09.2005 | 175.000 |
| 470081 | 641726 | 145325 | 3 | 481933 | 404 | 01.09.2005 | 245.001 |
| 470083 | 634309 | 147993 | 2 | 483268 | 403 | 01.09.2005 | 420.000 |
| 470086 | 623497 | 151291 | 3 | 482063 | 402 | 01.09.2005 | 100.001 |
| 430646 | 633100 | 129736 | 1 | 481913 | 402 | 01.09.2005 | 235.001 |
| 430647 | 632556 | 130926 | 1 | 481816 | 402 | 01.09.2005 | 100.000 |
| 430849 | 643434 | 132048 | 1 | 482118 | 404 | 01.09.2005 | 140.001 |
| 470089 | 630734 | 142147 | 2 | 481516 | 402 | 01.09.2005 | 350.001 |
| 470090 | 628562 | 145036 | 2 | 481497 | 402 | 01.09.2005 | 315.000 |
| 430651 | 627623 | 133741 | 1 | 484472 | 402 | 01.09.2005 | 70.000 |
| 430652 | 637201 | 130325 | 1 | 482042 | 403 | 01.09.2005 | 140.000 |
| 470094 | 626182 | 154728 | 3 | 480785 | 402 | 01.09.2005 | 250.000 |
| 470096 | 630695 | 156140 | 3 | 481801 | 402 | 01.09.2005 | 140.000 |
| 491108 | 626235 | 137742 | 2 | 481553 | 401 | 01.09.2005 | 210.000 |
| 470098 | 625318 | 151709 | 3 | 480707 | 402 | 01.09.2005 | 105.001 |
| 430853 | 645576 | 140579 | 1 | 480656 | 404 | 01.09.2005 | 105.000 |
| 470115 | 628241 | 140870 | 2 | 482376 | 402 | 01.09.2005 | 175.000 |
| 470117 | 628628 | 152672 | 3 | 480757 | 402 | 01.09.2005 | 70.000 |
| 470120 | 634782 | 143567 | 1 | 481884 | 403 | 01.09.2005 | 140.000 |
| 430658 | 632848 | 129706 | 1 | 480855 | 402 | 01.09.2005 | 105.001 |
| 470121 | 628502 | 151241 | 3 | 480706 | 402 | 01.09.2005 | 245.001 |

```
430660 633905 129121 1 480791 402 01.09.2005 400.000
470124 630574 142316 2 480722 402 01.09.2005 245.001
430661 623216 133120 1 481084 401 01.09.2005 100.000
470126 625162 153890 3 481817 402 01.09.2005 70.000
460368 617616 137814 2 480603 401 01.09.2005 150.000
470213 627822 155307 3 481186 402 01.09.2005 70.001
430863 642606 133835 1 481822 404 01.09.2005 111.001
470261 630801 155993 3 481148 402 01.09.2005 140.001
496263 631161 134140 1 480721 402 01.09.2005 175.000
470165 639872 149991 1 481185 404 01.09.2005 175.000
430681 630711 138836 1 481254 402 01.09.2005 175.001
430671 634679 129185 1 480641 402 01.09.2005 210.000
470142 637529 151615 3 481004 403 01.09.2005 105.000
470143 631251 153785 3 481049 402 01.09.2005 420.000
470144 625086 151558 3 481203 402 01.09.2005 210.001
470146 624711 151229 3 481393 402 01.09.2005 70.001
470148 638531 150296 1 481044 403 01.09.2005 105.000
470149 629110 142782 2 482369 402 01.09.2005 385.000
430861 649811 141531 1 481268 404 01.09.2005 105.001
430675 633294 128719 1 482083 402 01.09.2005 140.000
470152 636523 139481 1 481382 403 01.09.2005 240.000
430676 635293 130465 1 481781 403 01.09.2005 105.001
430680 631894 133478 1 482031 402 01.09.2005 280.000
430677 632734 133286 1 480999 402 01.09.2005 175.001
470155 637142 151580 3 481381 403 01.09.2005 245.000
430862 645817 134055 1 480965 404 01.09.2005 280.000
430864 647490 141591 1 481216 404 01.09.2005 420.001
470157 630938 153623 3 481398 402 01.09.2005 280.000
430865 644031 135819 1 480604 404 01.09.2005 210.000
470159 636560 147341 1 482355 403 01.09.2005 70.000
470160 631269 142002 2 482303 402 01.09.2005 105.001
470164 628900 142518 2 481713 402 01.09.2005 100.001
470166 638526 144565 1 481600 403 01.09.2005 385.000
470168 628871 152733 3 481400 402 01.09.2005 100.000
430684 633374 136358 1 480585 402 01.09.2005 385.000
470170 630612 142312 2 482384 402 01.09.2005 100.001
470172 635816 148576 1 481596 402 01.09.2005 105.000
470179 639359 147674 1 481911 403 01.09.2005 70.000
470180 624418 151353 3 480549 402 01.09.2005 105.001
470183 628918 143092 2 481572 402 01.09.2005 140.000
496640 627001 136931 1 482080 402 01.09.2005 140.001
470184 641133 145778 3 482076 404 01.09.2005 105.001
430871 645850 134236 1 481347 404 01.09.2005 140.000
490521 620714 136696 2 481266 401 01.09.2005 105.000
495627 632388 130538 1 480778 402 01.09.2005 65.000
470210 630225 156168 3 480864 402 01.09.2005 210.000
430698 632117 130122 1 482032 402 01.09.2005 140.000
496206 632938 130521 1 481234 402 01.09.2005 105.001
491112 622142 140804 2 481996 401 01.09.2005 175.000
430874 643084 133552 1 481222 404 01.09.2005 175.001
430696 632129 130662 1 481610 402 01.09.2005 70.000
470211 648663 155067 3 504122 404 01.09.2005 105.002
470259 641481 144195 3 487896 404 01.09.2005 70.001
460438 619090 138480 2 481462 401 01.09.2005 280.000
470209 638174 153819 3 481151 403 01.09.2005 245.001
```

```
470233 638918 152274 3 483269 403 01.09.2005 100.000
470234 635228 144782 1 480822 403 01.09.2005 175.000
430887 643248 135813 1 480811 404 01.09.2005 65.001
470235 638299 145130 1 481177 403 01.09.2005 450.001
430891 632806 129292 1 481734 402 01.09.2005 140.001
470231 628518 155261 3 482401 402 01.09.2005 105.000
430885 631562 130305 1 480602 402 01.09.2005 105.001
470230 640809 149927 1 481252 404 01.09.2005 105.000
430882 632660 129826 1 481038 402 01.09.2005 70.000
470227 628320 143215 2 482054 402 01.09.2005 105.001
430881 638723 135059 1 481740 403 01.09.2005 350.000
470224 628815 142030 2 480621 402 01.09.2005 105.001
430877 626362 133440 1 480584 402 01.09.2005 140.000
470218 642257 155195 3 480727 404 01.09.2005 460.000
470216 626045 151746 3 480541 402 01.09.2005 130.001
430893 637578 128218 1 481868 403 01.09.2005 105.000
470236 634457 157452 3 481211 402 01.09.2005 140.000
496648 630971 130363 1 481869 402 01.09.2005 70.001
496653 637334 133321 1 483092 403 01.09.2005 140.001
430900 644156 131811 1 481034 404 01.09.2005 175.000
430901 650547 138574 1 481150 404 01.09.2005 735.000
470260 638529 152259 3 481212 403 01.09.2005 250.001
460454 614267 139260 2 483279 401 01.09.2005 105.000
460450 616310 137758 2 483283 401 01.09.2005 100.002
430905 633010 129305 1 483289 402 01.09.2005 105.000
491042 623139 137733 483293 401 01.09.2005 70.000
460451 616870 137675 2 483306 401 01.09.2005 315.000
430907 627520 133290 1 483313 402 01.09.2005 140.000
470262 641179 149864 1 483335 404 01.09.2005 175.000
470264 628846 143420 2 483336 402 01.09.2005 105.001
470265 630753 139927 1 483339 402 01.09.2005 70.000
430909 644194 137792 1 483343 404 01.09.2005 140.000
430910 637937 134538 1 483345 403 01.09.2005 70.000
470266 624233 151159 3 483348 402 01.09.2005 70.001
470267 635475 145932 1 483349 403 01.09.2005 385.000
470268 633476 157031 3 483351 402 01.09.2005 490.000
430911 638570 135212 1 483352 403 01.09.2005 210.000
430912 636401 144636 1 483354 403 01.09.2005 900.000
470271 627666 154998 3 483355 402 01.09.2005 140.000
430913 635945 130743 1 483359 403 01.09.2005 200.001
470276 629886 155213 3 483467 402 01.09.2005 70.000
430915 633578 137320 1 483468 402 01.09.2005 280.000
470282 638440 146583 1 483627 403 01.09.2005 140.000
470283 633963 151313 3 483629 403 01.09.2005 525.000
430918 644053 137184 1 483647 404 01.09.2005 105.001
470284 636767 143513 1 483649 403 01.09.2005 200.000
470287 628105 144147 2 483661 402 01.09.2005 71.001
470289 638388 153048 3 483660 403 01.09.2005 210.000
470270 631989 143963 2 484293 402 01.09.2005 105.001
470277 642099 153548 3 484300 404 01.09.2005 70.000
430867 638522 130412 1 481420 403 03.08.2005 200.000
460438 619090 138480 2 496350 401 03.10.2005 175.000
1160331 616420 138460 2 481732 401 05.09.2005 65.000
1103855 614510 139880 2 480952 401 05.09.2005 300.000
1064211 615150 135295 1 480985 401 05.09.2005 70.000
```

```
1101948 615970 139750 2 481192 401 05.09.2005 240.001
1103584 614840 139880 2 481168 401 05.09.2005 420.000
1114648 620360 139475 2 481559 401 05.09.2005 175.000
1046082 616120 137440 2 481002 401 05.09.2005 210.000
1117017 621435 141950 2 480639 401 05.09.2005 110.001
1141070 622765 131675 1 480649 401 05.09.2005 210.000
1051955 618905 136100 1 481991 401 05.09.2005 100.000
1015703 619210 133040 1 480723 401 05.09.2005 160.000
1052706 618390 135540 1 481930 401 05.09.2005 140.001
1066205 619195 136880 1 480724 401 05.09.2005 105.000
1160315 616250 138540 2 480591 401 05.09.2005 140.000
1138563 623860 132315 1 481589 401 05.09.2005 200.000
1160598 616150 138060 2 480613 401 05.09.2005 140.000
1160598 616150 138060 2 504139 401 05.09.2005 140.000
1105616 615410 138090 2 480607 401 05.09.2005 105.000
1054347 619130 135225 1 480927 401 05.09.2005 132.000
1116370 621790 139955 2 486081 401 05.09.2005 350.001
1064187 615470 135470 1 481803 401 05.09.2005 140.000
1122204 621540 139095 2 481810 401 05.09.2005 105.000
1107082 616236 139464 2 481670 401 05.09.2005 140.001
1052527 618190 135890 1 481092 401 05.09.2005 35.000
1139612 623469 132920 1 481808 401 05.09.2005 175.000
1104115 614480 139400 2 482026 401 05.09.2005 70.000
1018351 619670 134460 1 481483 401 05.09.2005 105.000
1107220 616540 139180 2 481129 401 05.09.2005 65.000
1538481 629920 155650 3 482115 402 05.09.2005 105.000
1018211 619805 134455 1 480726 401 05.09.2005 70.000
1105605 615420 138160 2 481660 401 05.09.2005 65.000
1105225 614800 138630 2 480877 401 05.09.2005 70.000
1107527 615760 138950 2 481434 401 05.09.2005 140.002
1101840 616130 139670 2 480817 401 05.09.2005 140.000
1160373 616430 138510 2 481493 401 05.09.2005 130.000
1118110 620750 138340 2 481465 401 05.09.2005 100.000
1541640 623409 153792 3 481404 402 05.09.2005 350.000
1107187 616050 139280 2 481080 401 05.09.2005 380.000
1145368 625582 130591 1 481042 401 05.09.2005 270.000
1145792 626007 131012 1 481518 401 05.09.2005 105.000
1043565 615720 135480 1 481488 401 05.09.2005 70.000
1107529 615710 138960 2 480991 401 05.09.2005 70.000
1109197 617170 138770 2 480958 401 05.09.2005 1.000.001
1066185 619178 136807 1 480979 401 05.09.2005 175.000
1119517 622572 140718 2 481587 401 05.09.2005 120.001
1032185 616035 132190 1 481242 401 05.09.2005 1.100.000
1132000 622960 138173 2 483124 401 05.09.2005 140.000
1047287 616821 137102 2 483119 401 05.09.2005 140.002
1107117 616280 139320 2 481133 401 05.09.2005 105.000
1101610 616303 139974 2 483125 401 05.09.2005 200.000
1160385 616330 138510 2 480743 401 05.09.2005 65.000
1140403 622490 132885 1 481198 401 05.09.2005 175.000
1203502 626050 132680 1 480713 402 05.09.2005 105.000
1160558 616339 138213 2 483728 401 05.09.2005 65.001
1047504 616670 137680 2 484331 401 05.09.2005 315.001
428002 616050 137930 2 481605 401 05.09.2005 245.000
430202 622850 133370 1 481144 401 05.09.2005 490.000
430202 622850 133370 1 488722 401 05.09.2005 140.000
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 430224 | 623530 | 134925 | 1 | 481513 | 401 | 05.09.2005 | 70.000 |
| 430250 | 622573 | 131813 | 1 | 482007 | 401 | 05.09.2005 | 315.000 |
| 430305 | 625930 | 130870 | 1 | 481692 | 401 | 05.09.2005 | 100.000 |
| 430446 | 626277 | 134013 | 1 | 481451 | 402 | 05.09.2005 | 140.000 |
| 430451 | 627850 | 132768 | 1 | 481367 | 402 | 05.09.2005 | 140.001 |
| 430538 | 627908 | 132352 | 1 | 480808 | 402 | 05.09.2005 | 100.001 |
| 430550 | 628764 | 132772 | 1 | 480867 | 402 | 05.09.2005 | 105.000 |
| 440613 | 616019 | 135060 | 1 | 481607 | 401 | 05.09.2005 | 70.000 |
| 440621 | 616383 | 135868 | 1 | 481103 | 401 | 05.09.2005 | 70.001 |
| 460216 | 617430 | 137310 | 2 | 482005 | 401 | 05.09.2005 | 175.000 |
| 460221 | 617290 | 137880 | 2 | 480750 | 401 | 05.09.2005 | 300.000 |
| 460224 | 616960 | 137810 | 2 | 481601 | 401 | 05.09.2005 | 150.000 |
| 460232 | 616690 | 137850 | 2 | 480492 | 401 | 05.09.2005 | 70.000 |
| 460242 | 616620 | 137380 | 2 | 481087 | 401 | 05.09.2005 | 106.000 |
| 460261 | 617550 | 137420 | 2 | 480980 | 401 | 05.09.2005 | 140.001 |
| 460290 | 617400 | 138300 | 2 | 481101 | 401 | 05.09.2005 | 320.000 |
| 460331 | 616760 | 137720 | 2 | 481190 | 401 | 05.09.2005 | 100.000 |
| 460341 | 617531 | 136785 | 1 | 480789 | 401 | 05.09.2005 | 1.025.000 |
| 460401 | 617454 | 138797 | 2 | 481595 | 401 | 05.09.2005 | 105.000 |
| 460405 | 616880 | 140280 | 2 | 481137 | 401 | 05.09.2005 | 210.000 |
| 460419 | 617300 | 138440 | 2 | 481549 | 401 | 05.09.2005 | 490.000 |
| 460434 | 618269 | 138388 | 2 | 481090 | 401 | 05.09.2005 | 65.000 |
| 470206 | 627660 | 151530 | 2 | 480783 | 402 | 05.09.2005 | 145.000 |
| 470530 | 624736 | 151432 | 2 | 481544 | 402 | 05.09.2005 | 105.000 |
| 470530 | 624736 | 151432 | 2 | 509753 | 402 | 05.09.2005 | 105.000 |
| 470540 | 627077 | 151825 | 2 | 482073 | 402 | 05.09.2005 | 140.001 |
| 470540 | 627077 | 151825 | 2 | 511877 | 402 | 05.09.2005 | 140.001 |
| 470544 | 626489 | 154864 | 3 | 480943 | 402 | 05.09.2005 | 150.000 |
| 470552 | 631116 | 153458 | 3 | 480551 | 402 | 05.09.2005 | 245.000 |
| 470588 | 628617 | 154250 | 3 | 480873 | 402 | 05.09.2005 | 455.000 |
| 470588 | 628617 | 154250 | 3 | 509764 | 402 | 05.09.2005 | 455.000 |
| 470602 | 648640 | 150601 | 3 | 506638 | 404 | 05.09.2005 | 72.000 |
| 470654 | 626499 | 154855 | 3 | 481431 | 402 | 05.09.2005 | 70.000 |
| 470668 | 626426 | 154804 | 3 | 481458 | 402 | 05.09.2005 | 105.000 |
| 470671 | 626586 | 152159 | 2 | 480623 | 402 | 05.09.2005 | 385.000 |
| 470711 | 631111 | 152137 | 3 | 480787 | 402 | 05.09.2005 | 175.000 |
| 470786 | 628461 | 154582 | 3 | 481243 | 402 | 05.09.2005 | 280.000 |
| 470786 | 628461 | 154582 | 3 | 509765 | 402 | 05.09.2005 | 280.000 |
| 470796 | 627568 | 155036 | 3 | 481522 | 402 | 05.09.2005 | 70.001 |
| 490013 | 620270 | 134890 | 1 | 481459 | 401 | 05.09.2005 | 250.000 |
| 490411 | 618800 | 137358 | 1 | 481388 | 401 | 05.09.2005 | 140.000 |
| 490413 | 618273 | 135789 | 1 | 481099 | 401 | 05.09.2005 | 140.000 |
| 490421 | 620982 | 136878 | 2 | 482002 | 401 | 05.09.2005 | 100.000 |
| 490422 | 619525 | 136855 | 1 | 481720 | 401 | 05.09.2005 | 100.000 |
| 490437 | 618580 | 136880 | 1 | 481201 | 401 | 05.09.2005 | 35.000 |
| 490450 | 620564 | 133877 | 1 | 481463 | 401 | 05.09.2005 | 70.000 |
| 490460 | 618290 | 138136 | 2 | 480990 | 401 | 05.09.2005 | 150.000 |
| 490463 | 619682 | 137968 | 2 | 480949 | 401 | 05.09.2005 | 100.000 |
| 490471 | 618532 | 135883 | 1 | 480754 | 401 | 05.09.2005 | 70.000 |
| 490474 | 620950 | 136654 | 2 | 481131 | 401 | 05.09.2005 | 455.001 |
| 490496 | 618553 | 135330 | 1 | 481130 | 401 | 05.09.2005 | 140.000 |
| 490499 | 619435 | 136934 | 1 | 481368 | 401 | 05.09.2005 | 70.000 |
| 490501 | 618534 | 136326 | 1 | 481918 | 401 | 05.09.2005 | 100.000 |
| 491004 | 624430 | 137800 | 2 | 481096 | 401 | 05.09.2005 | 210.000 |
| 491008 | 621880 | 136885 | 2 | 480923 | 401 | 05.09.2005 | 140.001 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 491031 | 625256 | 138165 | 2 | 480705 | 401 | 05.09.2005 | 235.000 |
| 491041 | 622718 | 137759 | 2 | 480634 | 401 | 05.09.2005 | 210.001 |
| 491094 | 620385 | 139266 | 2 | 481511 | 401 | 05.09.2005 | 245.000 |
| 491096 | 622770 | 137804 | 2 | 481491 | 401 | 05.09.2005 | 175.000 |
| 470816 | 624950 | 151617 | 2 | 481075 | 402 | 05.09.2005 | 105.000 |
| 470816 | 624950 | 151617 | 2 | 509755 | 402 | 05.09.2005 | 105.000 |
| 470846 | 629045 | 152632 | 3 | 480531 | 402 | 05.09.2005 | 350.000 |
| 470846 | 629045 | 152632 | 3 | 509767 | 402 | 05.09.2005 | 350.000 |
| 470847 | 624816 | 151541 | 2 | 480631 | 402 | 05.09.2005 | 70.000 |
| 470847 | 624816 | 151541 | 2 | 509756 | 402 | 05.09.2005 | 70.000 |
| 460346 | 617797 | 137900 | 2 | 489920 | 401 | 05.09.2005 | 105.000 |
| 470014 | 624967 | 153842 | 3 | 482352 | 402 | 05.09.2005 | 175.000 |
| 490448 | 621467 | 137754 | 2 | 481189 | 401 | 05.09.2005 | 100.000 |
| 460238 | 617090 | 138430 | 2 | 481183 | 401 | 05.09.2005 | 70.000 |
| 470924 | 623438 | 143771 | 2 | 480595 | 402 | 05.09.2005 | 245.001 |
| 460436 | 618759 | 138982 | 2 | 481355 | 401 | 05.09.2005 | 130.000 |
| 470933 | 626796 | 151919 | 2 | 481410 | 402 | 05.09.2005 | 70.001 |
| 470933 | 626796 | 151919 | 2 | 511871 | 402 | 05.09.2005 | 70.002 |
| 470935 | 624256 | 154336 | 3 | 481149 | 402 | 05.09.2005 | 245.001 |
| 470935 | 624256 | 154336 | 3 | 492864 | 402 | 05.09.2005 | 245.000 |
| 470936 | 630387 | 155285 | 3 | 481541 | 402 | 05.09.2005 | 245.000 |
| 470936 | 630387 | 155285 | 3 | 509763 | 402 | 05.09.2005 | 245.000 |
| 440623 | 614834 | 131311 | 1 | 481498 | 401 | 05.09.2005 | 175.000 |
| 470948 | 626580 | 154917 | 3 | 480939 | 402 | 05.09.2005 | 140.000 |
| 470948 | 626580 | 154917 | 3 | 509822 | 402 | 05.09.2005 | 140.000 |
| 430509 | 628692 | 132669 | 1 | 481050 | 402 | 05.09.2005 | 315.000 |
| 470969 | 624824 | 154522 | 3 | 481665 | 402 | 05.09.2005 | 140.000 |
| 470976 | 626933 | 152334 | 2 | 481384 | 402 | 05.09.2005 | 140.000 |
| 470976 | 626933 | 152334 | 2 | 509759 | 402 | 05.09.2005 | 140.000 |
| 470980 | 629088 | 152641 | 3 | 480825 | 402 | 05.09.2005 | 100.000 |
| 460361 | 616065 | 138118 | 2 | 480749 | 401 | 05.09.2005 | 140.000 |
| 460366 | 614582 | 139000 | 2 | 481564 | 401 | 05.09.2005 | 140.001 |
| 430412 | 628862 | 132699 | 1 | 480651 | 402 | 05.09.2005 | 140.001 |
| 430644 | 625896 | 133251 | 1 | 482364 | 402 | 05.09.2005 | 140.000 |
| 430645 | 626979 | 135400 | 1 | 481163 | 402 | 05.09.2005 | 315.000 |
| 470080 | 629804 | 153410 | 3 | 480774 | 402 | 05.09.2005 | 175.000 |
| 430651 | 627623 | 133741 | 1 | 484472 | 402 | 05.09.2005 | 385.000 |
| 470093 | 621299 | 143116 | 2 | 481727 | 401 | 05.09.2005 | 105.001 |
| 470098 | 625318 | 151709 | 2 | 480707 | 402 | 05.09.2005 | 105.000 |
| 470098 | 625318 | 151709 | 2 | 509754 | 402 | 05.09.2005 | 105.000 |
| 460368 | 617616 | 137814 | 2 | 480603 | 401 | 05.09.2005 | 150.000 |
| 460368 | 617616 | 137814 | 2 | 507010 | 401 | 05.09.2005 | 145.000 |
| 470261 | 630801 | 155993 | 3 | 481148 | 402 | 05.09.2005 | 245.001 |
| 470261 | 630801 | 155993 | 3 | 496732 | 402 | 05.09.2005 | 245.000 |
| 460377 | 618265 | 139644 | 2 | 481508 | 401 | 05.09.2005 | 65.000 |
| 430672 | 627856 | 133621 | 1 | 481342 | 402 | 05.09.2005 | 210.001 |
| 470146 | 624711 | 151229 | 2 | 481393 | 402 | 05.09.2005 | 280.001 |
| 470146 | 624711 | 151229 | 2 | 509750 | 402 | 05.09.2005 | 280.000 |
| 470153 | 626955 | 151989 | 2 | 481919 | 402 | 05.09.2005 | 140.000 |
| 470153 | 626955 | 151989 | 2 | 511866 | 402 | 05.09.2005 | 140.001 |
| 470168 | 628871 | 152733 | 3 | 481400 | 402 | 05.09.2005 | 265.000 |
| 470168 | 628871 | 152733 | 3 | 509766 | 402 | 05.09.2005 | 280.000 |
| 470180 | 624418 | 151353 | 2 | 480549 | 402 | 05.09.2005 | 175.000 |
| 470180 | 624418 | 151353 | 2 | 509751 | 402 | 05.09.2005 | 175.000 |
| 490521 | 620714 | 136696 | 2 | 481266 | 401 | 05.09.2005 | 105.000 |

```
491112 622142 140804 2 486080 401 05.09.2005 244.000
491112 622142 140804 2 511314 401 05.09.2005 35.000
460442 616156 137880 2 480576 401 05.09.2005 70.000
460442 616156 137880 2 504131 401 05.09.2005 150.000
430876 627494 133356 1 481179 402 05.09.2005 70.001
490522 618489 137088 1 481867 401 05.09.2005 210.000
470251 626480 146662 2 506987 402 05.09.2005 105.001
460450 616310 137758 2 483283 401 05.09.2005 100.000
460451 616870 137675 2 483306 401 05.09.2005 245.000
460453 619000 138580 2 483311 401 05.09.2005 65.000
470271 627666 154998 3 483355 402 05.09.2005 175.000
470291 626657 152056 3 483659 402 05.09.2005 65.001
470291 626657 152056 3 509760 402 05.09.2005 70.000
470272 626720 152070 2 484302 402 05.09.2005 385.001
470272 626720 152070 2 509758 402 05.09.2005 385.000
470423 627340 155320 3 496740 402 05.10.2005 70.000
491081 620687 137779 2 496817 401 05.10.2005 315.000
470950 626662 154923 3 496729 402 05.10.2005 70.000
428022 616230 137870 2 509894 401 05.11.2005 105.000
470206 627660 151530 2 511873 402 06.12.2005 140.000
470552 631116 153458 3 511894 402 06.12.2005 245.000
470711 631111 152137 3 511884 402 06.12.2005 175.000
470980 629088 152641 3 511882 402 06.12.2005 100.000
470080 629804 153410 3 511889 402 06.12.2005 175.000
1105616 615410 138090 2 504149 401 07.11.2005 105.000
470671 626586 152159 2 513324 402 07.12.2005 210.000
428022 616230 137870 2 481896 401 10.09.2005 105.000
470996 630985 156410 3 498162 402 11.10.2005 140.000
460224 616960 137810 2 505342 401 11.11.2005 150.000
460401 617454 138797 2 505627 401 11.11.2005 245.000
430640 628049 141106 2 498714 402 13.10.2005 70.000
460341 617531 136785 2 505739 401 14.11.2005 105.000
470183 628918 143092 2 499257 402 15.10.2005 280.000
470264 628846 143420 2 499258 402 16.10.2005 105.000
470014 624967 153842 3 506641 402 16.11.2005 175.000
460341 617531 136785 2 499997 401 18.10.2005 525.000
470845 637356 152355 3 500198 403 19.10.2005 70.000
470142 637529 151615 3 500187 403 19.10.2005 105.000
1538481 629920 155650 3 507835 402 22.11.2005 105.000
1541640 623409 153792 3 507821 402 22.11.2005 350.000
470544 626489 154864 3 507827 402 22.11.2005 140.000
470654 626499 154855 3 507829 402 22.11.2005 70.000
470796 627568 155036 3 507832 402 22.11.2005 70.000
470969 624824 154522 3 507824 402 22.11.2005 140.000
470261 630801 155993 3 507840 402 22.11.2005 245.000
460242 616620 137380 2 508557 401 24.11.2005 315.000
1047080 616940 137590 2 494223 401 25.09.2005 70.000
460451 616870 137675 2 508818 401 25.11.2005 70.001
460443 619130 139027 1 501881 401 26.10.2005 65.001
1109197 617170 138770 2 494789 401 28.09.2005 175.000
470540 627077 151825 3 494854 402 28.09.2005 140.000
470935 624256 154336 3 509113 402 28.11.2005 245.000
470646 636351 154799 3 502563 403 29.10.2005 70.000
470908 638076 154091 3 502564 403 29.10.2005 140.000
470155 637142 151580 3 502562 403 29.10.2005 140.000
```

```
470668 626426 154804 3 509518 402 29.11.2005 105.000
470671 626586 152159 2 509757 402 29.11.2005 180.000
460237 617010 137520 2 502566 401 30.10.2005 70.000
1101610 616303 139974 2 488544 401 31.08.2005 200.000
1101610 616303 139974 2 488544 401 31.08.2005 200.000
470298 633751 151442 3 502625 403 31.10.2005 105.000
```

Appendix B

# Additional Results

## B.1 Results from the Parameter Tuning

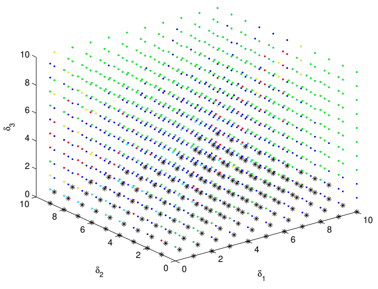
In this section results from the parameter tuning are presented. the colours represents the value of the cost function as described in section 7.3. In section B.1.1 results from the parametertuning of the construction heuristic is presented. And in B.1.2, results from the parameter tuning of the metaheuristic simulated annealing is presented.

### B.1.1 The Nearest Neighbour Construction Heuristic

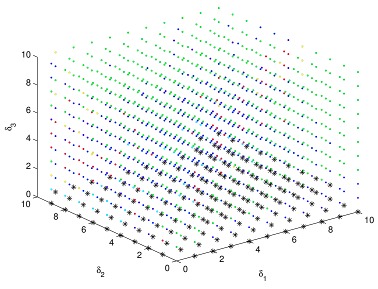
### B.1.2 Simulated Annealing

## B.2 Double Visits in the VRP approach

Some routes obtained in the VRP approach are presented here. The set of customers is $\mathcal{C}_d$, that is the customers which should be serviced twice. The notation is the same as in section 2.5.1. The initial three routes are seen in

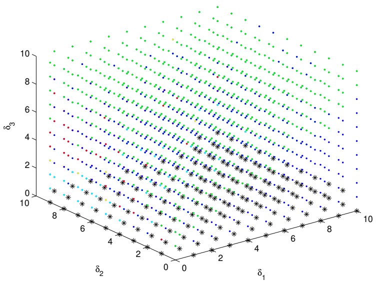


(a) Parametertuning for p370b

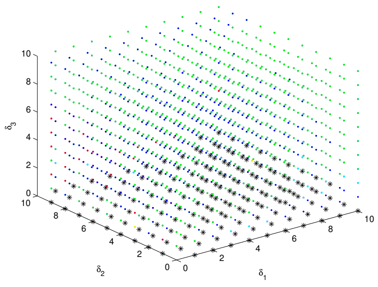


(b) Parametertuning for p420a

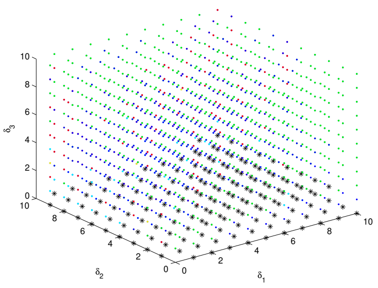
Figure B.1: Results of parametertuning on the data set p370b and p420a for the nearest neighbour construction heuristic.



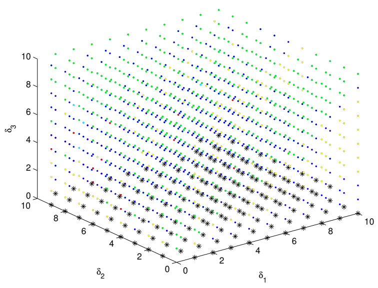
(a) Parametertuning for p420b



(b) Parametertuning for p470a

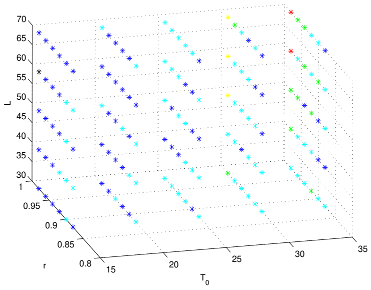


(c) Parametertuning for p470b

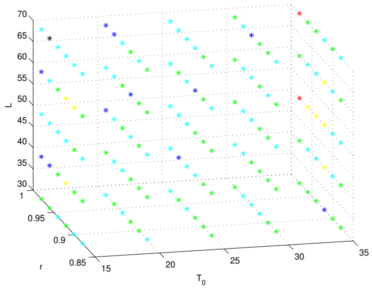


(d) Parametertuning for p5720

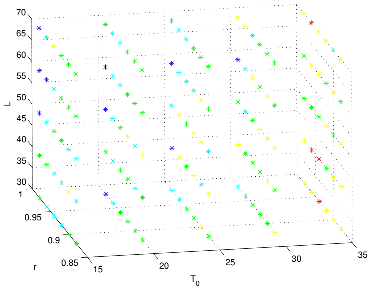
Figure B.2: Results of parametertuning on the data set p420a, p420b, p470a, p470b, p570 for the nearest neighbour construction heuristic.



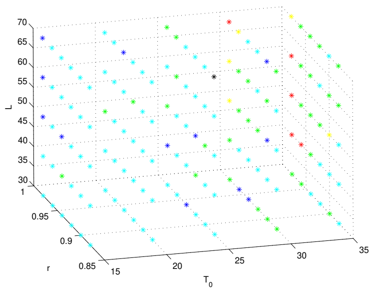

(a) Parametertuning for p370b




(b) Parametertuning for p420b




(c) Parametertuning for p470a




(d) Parametertuning for p570

Figure B.3: Results of parametertuning on the data set p370b, p420b, p470a and p570 for simulated annealing. Best solutions are obtained for low $T_0$

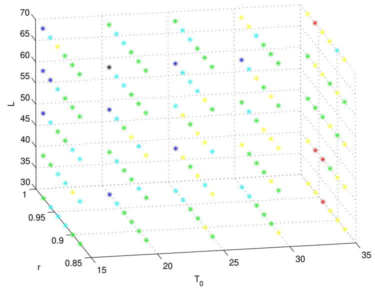



Figure B.4: Parameter tuning for simulated annealing on the data set p370a.

table B.1, B.2 and B.3. The routes after the two-opt procedure are seen in figure B.4, B.5 and B.6.

It is noted, that the two double visits for one customer is not scheduled imediately after each other. These are the visits $24'_0$ and $24'_1$. The reason is, that these visits are placed in two different groups, in the first part of the two phase method. The situation is illustrated on figure B.5.

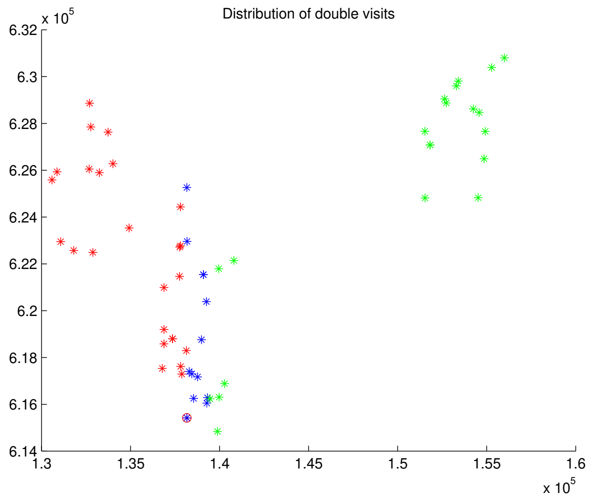



Figure B.5: The customers with double visits. The colours corresp0nds to the groups, defined in the first phase of the two phase method. For one customer the visits are placed in two different groups, the position of this customer is depicted by both a star and a circle, in colours corresponding to the two groups.

| Customer | UTM east | UTM north | demand |
|---|---|---|---|
| $0_0$ | 136785.0 | 617531.0 | 175000.0 |
| $0_1$ | 136785.0 | 617531.0 | 1025000.0 |
| $1_0$ | 137814.0 | 617616.0 | 150000.0 |
| $1_1$ | 137814.0 | 617616.0 | 150000.0 |
| $2_0$ | 137880.0 | 617290.0 | 300000.0 |
| $2_1$ | 137880.0 | 617290.0 | 130000.0 |
| $3_0$ | 138136.0 | 618290.0 | 150000.0 |
| $3_1$ | 138136.0 | 618290.0 | 150001.0 |
| $4_0$ | 137358.0 | 618800.0 | 140000.0 |
| $4_1$ | 137358.0 | 618800.0 | 70000.0 |
| $5_0$ | 136880.0 | 618580.0 | 35000.0 |
| $5_1$ | 136880.0 | 618580.0 | 70000.0 |
| $6_0$ | 136880.0 | 619195.0 | 105000.0 |
| $6_1$ | 136880.0 | 619195.0 | 105000.0 |
| $7_0$ | 136878.0 | 620982.0 | 100000.0 |
| $7_1$ | 136878.0 | 620982.0 | 100001.0 |
| $8_0$ | 137754.0 | 621467.0 | 100000.0 |
| $8_1$ | 137754.0 | 621467.0 | 100000.0 |
| $9_0$ | 137759.0 | 622718.0 | 210001.0 |
| $9_1$ | 137759.0 | 622718.0 | 175000.0 |
| $10_0$ | 137804.0 | 622770.0 | 150000.0 |
| $10_1$ | 137804.0 | 622770.0 | 150000.0 |
| $11_0$ | 137800.0 | 624430.0 | 210000.0 |
| $11_1$ | 137800.0 | 624430.0 | 105000.0 |
| $12_0$ | 134925.0 | 623530.0 | 70000.0 |
| $12_1$ | 134925.0 | 623530.0 | 210000.0 |
| $13_0$ | 132885.0 | 622490.0 | 175000.0 |
| $13_1$ | 132885.0 | 622490.0 | 175000.0 |
| $14_0$ | 131813.0 | 622573.0 | 315000.0 |
| $14_1$ | 131813.0 | 622573.0 | 140000.0 |
| $15_0$ | 131080.0 | 622950.0 | 175001.0 |
| $15_1$ | 131080.0 | 622950.0 | 175000.0 |
| $16_0$ | 130591.0 | 625582.0 | 270000.0 |
| $16_1$ | 130591.0 | 625582.0 | 135000.0 |
| $17_0$ | 130870.0 | 625930.0 | 100000.0 |
| $17_1$ | 130870.0 | 625930.0 | 130000.0 |
| $18_0$ | 132680.0 | 626050.0 | 70000.0 |
| $18_1$ | 132680.0 | 626050.0 | 70000.0 |
| $19_0$ | 133251.0 | 625896.0 | 140000.0 |
| $19_1$ | 133251.0 | 625896.0 | 105000.0 |
| $20_0$ | 134013.0 | 626277.0 | 140000.0 |
| $20_1$ | 134013.0 | 626277.0 | 70000.0 |
| $21_0$ | 133741.0 | 627623.0 | 315000.0 |
| $21_1$ | 133741.0 | 627623.0 | 70000.0 |
| $22_0$ | 132768.0 | 627850.0 | 140001.0 |
| $22_1$ | 132768.0 | 627850.0 | 105000.0 |
| $23_0$ | 132699.0 | 628862.0 | 140000.0 |
| $23_1$ | 132699.0 | 628862.0 | 70000.0 |
| $24'_0$ | 138160.0 | 615420.0 | 65000.0 |
| $25_0$ | 136785.0 | 617531.0 | 175000.0 |

Table B.1: Initial Route 1

| Customer | UTM east | UTM north | demand |
|---|---|---|---|
| $0_0$ | 138770.0 | 617170.0 | 1000001.0 |
| $0_1$ | 138770.0 | 617170.0 | 600000.0 |
| $1_0$ | 138770.0 | 617170.0 | 350000.0 |
| $1_1$ | 138770.0 | 617170.0 | 600000.0 |
| $2_0$ | 138770.0 | 617170.0 | 350000.0 |
| $2_1$ | 138770.0 | 617170.0 | 1000001.0 |
| $3_0$ | 138440.0 | 617300.0 | 490000.0 |
| $3_1$ | 138440.0 | 617300.0 | 140000.0 |
| $4_0$ | 138300.0 | 617400.0 | 320000.0 |
| $4_1$ | 138300.0 | 617400.0 | 150000.0 |
| $5_0$ | 138540.0 | 616250.0 | 140000.0 |
| $5_1$ | 138540.0 | 616250.0 | 70000.0 |
| $6_0$ | 139280.0 | 616050.0 | 380001.0 |
| $6_1$ | 139280.0 | 616050.0 | 210000.0 |
| $7_0$ | 139320.0 | 616280.0 | 105000.0 |
| $7_1$ | 139320.0 | 616280.0 | 70000.0 |
| $24'_1$ | 138160.0 | 615420.0 | 300000.0 |
| $8_0$ | 138982.0 | 618759.0 | 130000.0 |
| $8_1$ | 138982.0 | 618759.0 | 130000.0 |
| $9_0$ | 139266.0 | 620385.0 | 245000.0 |
| $9_1$ | 139266.0 | 620385.0 | 141001.0 |
| $10_0$ | 139095.0 | 621540.0 | 105000.0 |
| $10_1$ | 139095.0 | 621540.0 | 105000.0 |
| $11_0$ | 138173.0 | 622960.0 | 105000.0 |
| $11_1$ | 138173.0 | 622960.0 | 70000.0 |
| $12_0$ | 138165.0 | 625256.0 | 235000.0 |
| $12_1$ | 138165.0 | 625256.0 | 165000.0 |
| $13_0$ | 138770.0 | 617170.0 | 1000001.0 |

Table B.2: Initial Route number 2

| Customer | UTM east | UTM north | demand |
|---|---|---|---|
| $0_0$ | 154522.0 | 624824.0 | 140000.0 |
| $0_1$ | 154522.0 | 624824.0 | 105001.0 |
| $1_0$ | 154864.0 | 626489.0 | 150000.0 |
| $1_1$ | 154864.0 | 626489.0 | 150000.0 |
| $2_0$ | 154919.0 | 627654.0 | 175000.0 |
| $2_1$ | 154919.0 | 627654.0 | 140000.0 |
| $3_0$ | 154582.0 | 628461.0 | 280000.0 |
| $3_1$ | 154582.0 | 628461.0 | 210000.0 |
| $4_0$ | 154250.0 | 628617.0 | 455000.0 |
| $4_1$ | 154250.0 | 628617.0 | 245000.0 |
| $5_0$ | 153295.0 | 629602.0 | 175001.0 |
| $5_1$ | 153295.0 | 629602.0 | 70001.0 |
| $6_0$ | 153410.0 | 629804.0 | 175000.0 |
| $6_1$ | 153410.0 | 629804.0 | 175000.0 |
| $7_0$ | 152632.0 | 629045.0 | 350000.0 |
| $7_1$ | 152632.0 | 629045.0 | 210000.0 |
| $8_0$ | 152733.0 | 628871.0 | 265000.0 |
| $8_1$ | 152733.0 | 628871.0 | 100000.0 |
| $9_0$ | 151530.0 | 627660.0 | 145000.0 |
| $9_1$ | 151530.0 | 627660.0 | 145001.0 |
| $10_0$ | 151825.0 | 627077.0 | 140000.0 |
| $10_1$ | 151825.0 | 627077.0 | 140001.0 |
| $11_0$ | 151541.0 | 624816.0 | 70000.0 |
| $11_1$ | 151541.0 | 624816.0 | 70001.0 |
| $12_0$ | 155285.0 | 630387.0 | 245000.0 |
| $12_1$ | 155285.0 | 630387.0 | 105000.0 |
| $13_0$ | 155993.0 | 630801.0 | 245001.0 |
| $13_1$ | 155993.0 | 630801.0 | 140001.0 |
| $14_0$ | 140804.0 | 622142.0 | 280000.0 |
| $14_1$ | 140804.0 | 622142.0 | 175000.0 |
| $15_0$ | 139955.0 | 621790.0 | 315000.0 |
| $15_1$ | 139955.0 | 621790.0 | 315000.0 |
| $16_0$ | 140280.0 | 616880.0 | 210000.0 |
| $16_1$ | 140280.0 | 616880.0 | 210000.0 |
| $17_0$ | 139974.0 | 616303.0 | 200000.0 |
| $17_1$ | 139974.0 | 616303.0 | 200000.0 |
| $18_0$ | 139464.0 | 616236.0 | 140000.0 |
| $18_1$ | 139464.0 | 616236.0 | 140000.0 |
| $19_0$ | 139880.0 | 614840.0 | 350000.0 |
| $19_1$ | 139880.0 | 614840.0 | 130000.0 |
| $20_0$ | 154522.0 | 624824.0 | 140000.0 |

Table B.3: Initial route number 3

| Customer | UTM east | UTM north | demand |
|---|---|---|---|
| $0_0$ | 136785.0 | 617531.0 | 175000.0 |
| $0_1$ | 136785.0 | 617531.0 | 1025000.0 |
| $1_0$ | 137814.0 | 617616.0 | 150000.0 |
| $1_1$ | 137814.0 | 617616.0 | 150000.0 |
| $2_0$ | 137880.0 | 617290.0 | 300000.0 |
| $2_1$ | 137880.0 | 617290.0 | 130000.0 |
| $3_0$ | 138136.0 | 618290.0 | 150000.0 |
| $3_1$ | 138136.0 | 618290.0 | 150001.0 |
| $4_0$ | 137358.0 | 618800.0 | 140000.0 |
| $4_1$ | 137358.0 | 618800.0 | 70000.0 |
| $5_0$ | 136880.0 | 618580.0 | 35000.0 |
| $5_1$ | 136880.0 | 618580.0 | 70000.0 |
| $6_0$ | 136880.0 | 619195.0 | 105000.0 |
| $6_1$ | 136880.0 | 619195.0 | 105000.0 |
| $7_0$ | 136878.0 | 620982.0 | 100000.0 |
| $7_1$ | 136878.0 | 620982.0 | 100001.0 |
| $8_0$ | 137754.0 | 621467.0 | 100000.0 |
| $8_1$ | 137754.0 | 621467.0 | 100000.0 |
| $9_0$ | 137759.0 | 622718.0 | 210001.0 |
| $9_1$ | 137759.0 | 622718.0 | 175000.0 |
| $10_0$ | 137804.0 | 622770.0 | 150000.0 |
| $10_1$ | 137804.0 | 622770.0 | 150000.0 |
| $11_0$ | 137800.0 | 624430.0 | 210000.0 |
| $11_1$ | 137800.0 | 624430.0 | 105000.0 |
| $12_0$ | 134925.0 | 623530.0 | 70000.0 |
| $12_1$ | 134925.0 | 623530.0 | 210000.0 |
| $13_0$ | 132885.0 | 622490.0 | 175000.0 |
| $13_1$ | 132885.0 | 622490.0 | 175000.0 |
| $14_0$ | 131813.0 | 622573.0 | 315000.0 |
| $14_1$ | 131813.0 | 622573.0 | 140000.0 |
| $15_0$ | 131080.0 | 622950.0 | 175001.0 |
| $15_1$ | 131080.0 | 622950.0 | 175000.0 |
| $16_0$ | 130591.0 | 625582.0 | 270000.0 |
| $16_1$ | 130591.0 | 625582.0 | 135000.0 |
| $17_0$ | 130870.0 | 625930.0 | 100000.0 |
| $17_1$ | 130870.0 | 625930.0 | 130000.0 |
| $18_0$ | 132680.0 | 626050.0 | 70000.0 |
| $18_1$ | 132680.0 | 626050.0 | 70000.0 |
| $19_0$ | 133251.0 | 625896.0 | 140000.0 |
| $19_1$ | 133251.0 | 625896.0 | 105000.0 |
| $20_0$ | 134013.0 | 626277.0 | 140000.0 |
| $20_1$ | 134013.0 | 626277.0 | 70000.0 |
| $21_0$ | 133741.0 | 627623.0 | 315000.0 |
| $21_1$ | 133741.0 | 627623.0 | 70000.0 |
| $22_0$ | 132768.0 | 627850.0 | 140001.0 |
| $22_1$ | 132768.0 | 627850.0 | 105000.0 |
| $23_0$ | 132699.0 | 628862.0 | 140000.0 |
| $23_1$ | 132699.0 | 628862.0 | 70000.0 |
| $24'_0$ | 138160.0 | 615420.0 | 65000.0 |
| $25_0$ | 136785.0 | 617531.0 | 175000.0 |

Table B.4: Route number 1 after the two-opt procedure

| Customer | UTM east | UTM north | demand |
|---|---|---|---|
| $0_0$ | 138770.0 | 617170.0 | 1000001.0 |
| $0_1$ | 138770.0 | 617170.0 | 600000.0 |
| $1_0$ | 138770.0 | 617170.0 | 350000.0 |
| $1_1$ | 138770.0 | 617170.0 | 600000.0 |
| $2_0$ | 138770.0 | 617170.0 | 350000.0 |
| $2_1$ | 138770.0 | 617170.0 | 1000001.0 |
| $3_0$ | 138440.0 | 617300.0 | 490000.0 |
| $3_1$ | 138440.0 | 617300.0 | 140000.0 |
| $4_0$ | 138300.0 | 617400.0 | 320000.0 |
| $4_1$ | 138300.0 | 617400.0 | 150000.0 |
| $5_0$ | 138540.0 | 616250.0 | 140000.0 |
| $5_1$ | 138540.0 | 616250.0 | 70000.0 |
| $6_0$ | 139280.0 | 616050.0 | 380001.0 |
| $6_1$ | 139280.0 | 616050.0 | 210000.0 |
| $7_0$ | 139320.0 | 616280.0 | 105000.0 |
| $7_1$ | 139320.0 | 616280.0 | 70000.0 |
| $24'_1$ | 138160.0 | 615420.0 | 300000.0 |
| $8_0$ | 138982.0 | 618759.0 | 130000.0 |
| $8_1$ | 138982.0 | 618759.0 | 130000.0 |
| $9_0$ | 139266.0 | 620385.0 | 245000.0 |
| $9_1$ | 139266.0 | 620385.0 | 141001.0 |
| $10_0$ | 139095.0 | 621540.0 | 105000.0 |
| $10_1$ | 139095.0 | 621540.0 | 105000.0 |
| $11_0$ | 138173.0 | 622960.0 | 105000.0 |
| $11_1$ | 138173.0 | 622960.0 | 70000.0 |
| $12_0$ | 138165.0 | 625256.0 | 235000.0 |
| $12_1$ | 138165.0 | 625256.0 | 165000.0 |
| $13_0$ | 138770.0 | 617170.0 | 1000001.0 |

Table B.5: Route number 2 after the two-opt procedure

| Customer | UTM east | UTM north | demand |
|---|---|---|---|
| $0_0$ | 154522.0 | 624824.0 | 140000.0 |
| $0_1$ | 154522.0 | 624824.0 | 105001.0 |
| $1_0$ | 154864.0 | 626489.0 | 150000.0 |
| $1_1$ | 154864.0 | 626489.0 | 150000.0 |
| $2_0$ | 154919.0 | 627654.0 | 175000.0 |
| $2_1$ | 154919.0 | 627654.0 | 140000.0 |
| $3_0$ | 154582.0 | 628461.0 | 280000.0 |
| $3_1$ | 154582.0 | 628461.0 | 210000.0 |
| $4_0$ | 154250.0 | 628617.0 | 455000.0 |
| $4_1$ | 154250.0 | 628617.0 | 245000.0 |
| $5_0$ | 153295.0 | 629602.0 | 175001.0 |
| $5_1$ | 153295.0 | 629602.0 | 70001.0 |
| $6_0$ | 153410.0 | 629804.0 | 175000.0 |
| $6_1$ | 153410.0 | 629804.0 | 175000.0 |
| $7_0$ | 152632.0 | 629045.0 | 350000.0 |
| $7_1$ | 152632.0 | 629045.0 | 210000.0 |
| $8_0$ | 152733.0 | 628871.0 | 265000.0 |
| $8_1$ | 152733.0 | 628871.0 | 100000.0 |
| $9_0$ | 151530.0 | 627660.0 | 145000.0 |
| $9_1$ | 151530.0 | 627660.0 | 145001.0 |
| $10_0$ | 151825.0 | 627077.0 | 140000.0 |
| $10_1$ | 151825.0 | 627077.0 | 140001.0 |
| $11_0$ | 151541.0 | 624816.0 | 70000.0 |
| $11_1$ | 151541.0 | 624816.0 | 70001.0 |
| $12_0$ | 155285.0 | 630387.0 | 245000.0 |
| $12_1$ | 155285.0 | 630387.0 | 105000.0 |
| $13_0$ | 155993.0 | 630801.0 | 245001.0 |
| $13_1$ | 155993.0 | 630801.0 | 140001.0 |
| $14_0$ | 140804.0 | 622142.0 | 280000.0 |
| $14_1$ | 140804.0 | 622142.0 | 175000.0 |
| $15_0$ | 139955.0 | 621790.0 | 315000.0 |
| $15_1$ | 139955.0 | 621790.0 | 315000.0 |
| $16_0$ | 140280.0 | 616880.0 | 210000.0 |
| $16_1$ | 140280.0 | 616880.0 | 210000.0 |
| $17_0$ | 139974.0 | 616303.0 | 200000.0 |
| $17_1$ | 139974.0 | 616303.0 | 200000.0 |
| $18_0$ | 139464.0 | 616236.0 | 140000.0 |
| $18_1$ | 139464.0 | 616236.0 | 140000.0 |
| $19_0$ | 139880.0 | 614840.0 | 350000.0 |
| $19_1$ | 139880.0 | 614840.0 | 130000.0 |
| $20_0$ | 154522.0 | 624824.0 | 140000.0 |

Table B.6: Route number 3 after the two-opt procedure

Appendix C

# Source Code

## C.1 Programs for the VRP Formulation

### C.1.1 Data Structures

#### C.1.1.1 Customer.java

```
public class Customer {
    public double[] coordinates = new double[2];
    public double bulk;

    public Customer(double e, double n, double b){
coordinates[0] = e;
coordinates[1] = n;
bulk = b;
    }
    public double demand(){return bulk;}
    public double[] position(){return coordinates;}
    public String toStringTable(){
   String s = coordinates[0]+ " & " + coordinates[1] + " & " + bulk +
    " \\" + "\\ \n" ;
   return s;
    }
    public String toString() {
     String s = "position (" + coordinates[0] +","+ coordinates[1] + ")
```

```
    \t demand " + bulk + "\n";
    return s;
    }
    public String toStringX(){
String s = coordinates[0]+ " \t";
return s;
    }
    public String toStringY(){
String s = coordinates[1] +" \t";
return s;
    }
}
```

### C.1.1.2 Route.java

```
import java.util.*;

public class Route {
    public Vector v = new Vector();

    public Route(){};

    //Add customer to the end of theroute
    public boolean addCustomer(Customer c){return v.add(c);}
    // Add customer to the specified position of the routs
    public void addCustomerAt(int n,Customer c){
v.add(n,c);
    }
    public boolean removeCustomer(Customer c){return v.remove(c);}

    //Get customer from the route
    public Customer getCustomer(int i){return (Customer) v.get(i);}

    //Number of customers in the route
    public int routeSize(){return v.size();}

    //Remove all customers from the route
    public void empty(){v.clear();}

    //The length of the route
    public int pathLength(){
      int cost = 0;
      int l = v.size();

      for(int i=0; i<l; i++)
    {
Customer n = (Customer) v.get(i);
cost += n.demand();
    }
return cost;
}
```

```
    public String toString(){
String output = new String();
int l = v.size();
for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
  output += c.toString();
      }
  return output;
    }
    public String toStringTable(){
String output = new String();
int l = v.size();
for(int i=0; i<l; i++)
      {
int j = i/2;
int rem =i%2;
  Customer c = (Customer) v.get(i);
  output += "$"+j + "_" + rem + "$ & " + c.toStringTable();
      }
  return output;
    }
    public String toStringX(){
String output = new String();
int l = v.size();
for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
  output += c.toStringX();
      }
  return output;
    }
    public String toStringY(){
String output = new String();
int l = v.size();
for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
  output += c.toStringY();
      }
  return output;
    }
}
```

## C.1.2 Solution Methods

### C.1.2.1 FindSolution.java

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

```
public class FindSolution{
  public Data data;
  public FindSolution(String datafile){
System.out.println("clear all, close all");
data = new Data(datafile);
int vehicles = 3;
int n = 1;
Vector customersRaw = data.getData();
//Solves the problem in northern Sweeden

Vector cnorth = findNorthernCustomers(n, customersRaw);
Vector cdouble = new Vector();
Vector customersRaw1 = data.getData();
for(int j = 0; j<customersRaw1.size()-1;j++){
    for(int i = j+1; i<customersRaw1.size();i++){
Customer a = (Customer) customersRaw1.get(i);
Customer b = (Customer) customersRaw1.get(j);
if(identicalCustomer(a, b)){
     cdouble.add(a);
     cdouble.add(b);
}
    }
}
Vector sortedSouth = SortCustomers(cnorth);
DivideCustomersNorth(vehicles, sortedSouth);
  }

  public Vector findNorthernCustomers(int n, Vector v){
int northBorder = 640000;
int nNorth = 0;
int nSouth = 0;
Vector customerSouth = new Vector();
Vector customerNorth = new Vector();
Customer c;
int nline = v.size();
for(int i=0; i<nline; i++){
    c = (Customer) v.get(i);
    double north = c.position()[1];
    if(north>northBorder){
customerNorth.add(c);
nNorth++;
   }
     else{
customerSouth.add(c);
     }
}
if(n == 1){
    return customerNorth;
}
else{
    return customerSouth;
}
  }

    //Kunder bliver delt - 1. fase af 2-phase metoden
```

```
  public Vector SortCustomers(Vector customers){
int l = customers.size();
for (int index = 1; index < l; index++){
    Customer cos = (Customer) customers.get(index);
    customers.remove(index);
    double[] pos = cos.position();
    double east = pos[0];
    int sortPos = index-1;
    Customer temp = (Customer) customers.get(sortPos);
    //Shift larger values to the rigth
    while (sortPos > -1 && ...
    ((Customer) customers.get(sortPos)).position()[0] > east){
temp = (Customer) customers.get(sortPos);
sortPos--;
    }
    customers.add(sortPos+1,cos);
}
return customers;
  }
// Sorts customers after their x-coordinates. Returns Southernmost
// customer for each vehicle since this is needed in
// the DivideCustomersSouth function
   public Customer[] DivideCustomersNorth(int cars, Vector customers){
double totalDemand = 0;
int l = customers.size();
for (int index = 0; index < l; index++)
    {
Customer cosnew = (Customer) customers.get(index);
totalDemand = totalDemand + cosnew.demand();
    }
// Hvor meget bulk skal den enkelte bil presse
double vehicleDemand = totalDemand/cars;
// Kunderne bliver delt op efter oestlig beliggenhed
double usedDemand = 0;
int i = 0;
Route[] routes = new Route[cars];
Customer[] south = new Customer[cars];
for(int k = 0; k < cars; k++){
    Route route = new Route();
    while (usedDemand < vehicleDemand && i < l){
Customer c = (Customer) customers.get(i);
double dem = c.demand();
usedDemand = usedDemand + dem;
boolean b = route.addCustomer(c);
i++;
   }
    usedDemand = usedDemand - (k+1)*vehicleDemand;
    // Initiel lsn. i nordsverige
//Initial time
Date startTime = new Date();
    Route sortedRoute = nearestNeighbourNorth(route);
Date stopTime = new Date();
long time = stopTime.getTime()-startTime.getTime();
//System.out.println("NNtime =" + time);
    routes[k]=sortedRoute;
```

```
    String sx = sortedRoute.toStringX();
    String sy = sortedRoute.toStringY();
    String table = sortedRoute.toStringTable();
    double cost = totalDistance(sortedRoute);
    // Udprint af initielle ruter til Matlab
    System.out.println("CostnorthInit" + k + "= " + cost);
    System.out.println("X" + k + "= [" + sx + "];");
    System.out.println("Y" + k + "= [" + sy + "];");
    //System.out.println("Init" + k);
    //System.out.println(table);
    //     int z=0;
    // 2-opt procedure i NS
    Date startTime2opt = new Date();
    Route twoOptRoute = twoOpt(sortedRoute);
    Date stopTime2opt = new Date();
    long time2opt = stopTime2opt.getTime()-startTime2opt.getTime();
    //System.out.println("2opttime =" + time2opt);
    double costTwoOpt = totalDistance(twoOptRoute);
    String twooptx = twoOptRoute.toStringX();
    String twoopty = twoOptRoute.toStringY();
    String table2opt = sortedRoute.toStringTable();
    // Print af 2-opt lsn til Matlab
    System.out.println("CostnorthTwoOpt" + k + "= " + costTwoOpt);
    System.out.println("XTwoOpt" + k + "= [" + twooptx + "];");
    System.out.println("YTwoOpt" + k + "= [" + twoopty + "];");
    //System.out.println("twoopt" + k);
    //System.out.println(table2opt);
    south[k] = FindInitialCustomer(sortedRoute);
}
// Print til Matlabfil, kommando for at plotte initiel + 2-opt rute
System.out.println("figure");
System.out.println("title('Initial Solution Northern Sweden')");
System.out.println("plot(X0,Y0,'r-') \n hold on");
System.out.println("plot(X1,Y1,'b-') \n hold on");
System.out.println("plot(X2,Y2,'g-') \n hold on");
System.out.println("plot(X0(1),Y0(1),'r*') \n hold on");
System.out.println("plot(X1(1),Y1(1),'b*') \n hold on");
System.out.println("plot(X2(1),Y2(1),'g*')");
System.out.println("figure");
System.out.println("title('2-opt Solution Northern Sweden') \n hold on");
System.out.println("plot(XTwoOpt0,YTwoOpt0,'r-') \n hold on");
System.out.println("plot(XTwoOpt1,YTwoOpt1,'b-') \n hold on");
System.out.println("plot(XTwoOpt2,YTwoOpt2,'g-') \n hold on");
System.out.println("plot(XTwoOpt0(1),YTwoOpt0(1),'r*') \n hold on");
System.out.println("plot(XTwoOpt1(1),YTwoOpt1(1),'b*') \n hold on");
System.out.println("plot(XTwoOpt2(1),YTwoOpt2(1),'g*')");
// Den sydligste beliggende kunde i hver rute returneres
return south;
  }

  public void DivideCustomersSouth...
  (int cars, Vector customers, Customer[] init){
double totalDemand = 0;
int l = customers.size();
for (int index = 0; index < l; index++)
```

```
    {
Customer cosnew = (Customer) customers.get(index);
totalDemand = totalDemand + cosnew.demand();
    }
// Hvor meget bulk skal den enkelte bil presse
double vehicleDemand = totalDemand/cars;
// Kunderne bliver delt op efter oestlig beliggenhed
double usedDemand = 0;
int i = 0;
Route[] routes = new Route[cars];
for(int k = 0; k < cars; k++){
    Route route = new Route();
    while (usedDemand < vehicleDemand && i < l){
Customer c = (Customer) customers.get(i);
double dem = c.demand();
usedDemand = usedDemand + dem;
boolean b = route.addCustomer(c);
i++;
    }
    usedDemand = usedDemand - (k+1)*vehicleDemand;
    // Find startcustomer for the tour, i.e. southernmost
    // customer in Northern Sweeden. For each vehicle....
    Customer initial = init[k];
    // Beregn initiel lsn. i SS, vha. nearest neighbor
    Route sortedRoute = nearestNeighbourSouth(route,initial);
    routes[k]=sortedRoute;
    String sx = sortedRoute.toStringX();
    String sy = sortedRoute.toStringY();
    double cost = totalDistance(sortedRoute);
    // Udprint til Matlab - initiel lsn.
    System.out.println("Costsouth" + k + "= " + cost);
    System.out.println("Xs" + k + "= [" + sx + "];");
    System.out.println("Ys" + k + "= [" + sy + "];");
    // 2-opt i SS
    Route twoOptRoute = twoOptSouth(sortedRoute);
    double costTwoOpt = totalDistance(twoOptRoute);
    String twooptx = twoOptRoute.toStringX();
    String twoopty = twoOptRoute.toStringY();
    // Udprint til Matlab
    System.out.println("CostSouthTwoOpt" + k + "= " + costTwoOpt);
    System.out.println("XTwoOptSouth" + k + "= [" + twooptx + "];");
    System.out.println("YTwoOptSouth" + k + "= [" + twoopty + "];");
}
// Print til Matlabfil, der plotter initiel rute
System.out.println("figure");
System.out.println("title('Initial Solution Southern Sweden')");
System.out.println("hold on");
System.out.println("plot(Xs0,Ys0,'r-') \n hold on");
System.out.println("plot(Xs1,Ys1,'b-') \n hold on");
System.out.println("plot(Xs2,Ys2,'g-')");
System.out.println("figure");
System.out.println("title('2-opt Solution Southern Sweden')");
System.out.println("plot(XTwoOptSouth0,YTwoOptSouth0,'r-') \n hold on");
System.out.println("plot(XTwoOptSouth1,YTwoOptSouth1,'b-') \n hold on");
System.out.println("plot(XTwoOptSouth2,YTwoOptSouth2,'g-')");
```

```
    }
    //Find distance between 2 customers
    public double findDistance(Customer a, Customer b){
double[] posa = a.position();
double[] posb = b.position();
double difEast = Math.pow((posa[0]-posb[0]),2);
double difNorth = Math.pow((posa[1]-posb[1]),2);
double distance = Math.sqrt(difEast+difNorth);
return distance;
    }
    // Check if two customers are the same
    public boolean identicalCustomer(Customer a, Customer b){
double[] posa = a.position();
double[] posb = b.position();
if(posa[0]==posb[0] && posa[1]==posb[1]){
    return true;
}
else{
    return false;
}
  }

  // Find the southernmost customers in northern Sweden for each vehicle
  public Customer FindInitialCustomer(Route r){
int size = r.routeSize();
Customer cmin = r.getCustomer(0);
double ymin = cmin.position()[1];
for(int i=1; i < size; i++){
   Customer compare = r.getCustomer(i);
    double ycompare = cmin.position()[1];
    if(ycompare < ymin){
ymin = ycompare;
cmin = compare;
    }
}
return cmin;
  }

  // Solve the resulting TSP by the nearest neighbor heuristic
  // (i.e. 2. phase of 2-phase method)
  private Route nearestNeighbourNorth(Route r){
Route newRoute = new Route();
int size = r.routeSize();
int ran = (int) Math.round(Math.random()*(size-1));
Customer j = r.getCustomer(ran);
boolean b = newRoute.addCustomer(j);
boolean bool = r.removeCustomer(j);
size--;
int start = 0;
while(size > 1){
    Customer cmin = r.getCustomer(start);
    double distmin = findDistance(cmin, j);
    for(int i=1; i < size; i++){
Customer l = r.getCustomer(i);
double dist = findDistance(l,j);
```

```
if(dist<distmin){
    distmin = dist;
    cmin = l;
}
    }
    bool = r.removeCustomer(cmin);
    b = newRoute.addCustomer(cmin);
    j = cmin;
    size--;
}
Customer last = r.getCustomer(start);
b = newRoute.addCustomer(last);
bool = r.removeCustomer(last);
// Connect the last customer to the first customer
//Customer first = newRoute.getCustomer(start);
//b = newRoute.addCustomer(first);
return newRoute;
    }

  private Route nearestNeighbourSouth(Route r, Customer cstart){
Route newRoute = new Route();
int size = r.routeSize();
boolean b = newRoute.addCustomer(cstart);
boolean bool = r.removeCustomer(cstart);
size--;
int start = 0;
Customer j = cstart;
while(size > 1){
 Customer cmin = r.getCustomer(start);
 double distmin = findDistance(cmin, j);
 for(int i=1; i < size; i++){
 Customer l = r.getCustomer(i);
 double dist = findDistance(l,j);
if(dist<distmin){
    distmin = dist;
    cmin = l;
}
}
    bool = r.removeCustomer(cmin);
    b = newRoute.addCustomer(cmin);
    j = cmin;
    size--;
}
Customer last = r.getCustomer(start);
b = newRoute.addCustomer(last);
bool = r.removeCustomer(last);
// Coordinates for the depot
double dy = 619800;
double dx = 134200;
double dd = 0;
Customer depot = new Customer(dx,dy,dd);
// Connect the last customer to the depot
b = newRoute.addCustomer(depot);
return newRoute;
    }
```

```
    // Calculates the total distance at one particular route
    public double totalDistance(Route r){
int j = 0;
Customer cj = r.getCustomer(j);
int size = r.routeSize();
double cost = 0;
for(int i=1; i < size; i++){
    Customer ci = r.getCustomer(i);
    double dist = findDistance(cj,ci);
    cost=cost+dist;
    cj = ci;
}
return cost;
    }
    // Funktionen two opt, benytter 2-exchange
    // metode til ruten er 2-optimal
    // Funktionen kan benyttes i NS
    public Route twoOpt(Route r){
int size = r.routeSize();
Route nextr = new Route();

// Ny rute nextr, indeholder ikke slutknude
for(int k=0; k < size; k++){
    Customer c =  r.getCustomer(k);
    boolean b = nextr.addCustomer(c);
}
double objectfct = totalDistance(r);
for(int i=0; i <size; i++){
    for(int j=0; j <size; j++){
Route newRoute = twoExchange(i,j,nextr);
Customer first =  newRoute.getCustomer(0);
newRoute.addCustomer(first);
double compfct = totalDistance(newRoute);
if(compfct < objectfct){
    System.out.println("% i = " +i+" j = " +j);
    nextr = newRoute;
    objectfct = compfct;
    i=0;
    j=0;
}
    }
}
return nextr;
    }
    // Funktionen two opt, benytter 2-exchange metode
    // til ruten er 2-optimal
    // Funktionen kan benyttes i SS
    public Route twoOptSouth(Route r){
int size = r.routeSize();
Route nextr = new Route();
size--;
Customer start =  r.getCustomer(0);
// Ny rute nextr, indeholder hverken start knude eller depot
for(int k=1; k < size; k++){
```

```
    Customer c =  r.getCustomer(k);
    boolean b = nextr.addCustomer(c);
}
Customer depot =  r.getCustomer(size);
double objectfct = totalDistance(r);
for(int i=0; i <= size; i++){
    for(int j=0; j <= size; j++){
Route newRoute = twoExchange(i,j,nextr);
newRoute.addCustomerAt(0,start);
newRoute.addCustomer(depot);
double compfct = totalDistance(newRoute);
if(compfct < objectfct){
    System.out.println("% i = " +i+" j = " +j);
    nextr = newRoute;
    objectfct = compfct;
    i=0;
    j=0;
}
    }
}
return nextr;
    }

    public Route twoExchange(int i, int j, Route v){
int size = v.routeSize();
Route nextr = new Route();
// Ny rute next,
for(int k=0; k < size; k++){
    Customer c =  v.getCustomer(k);
    boolean b = nextr.addCustomer(c);
}
if(j<i || i<0 || j<1 || i>size-1 || j > size){
    return nextr;
}
else{
    i++;
    j--;
    while(i<j){
Customer ci = nextr.getCustomer(i);
Customer cj = nextr.getCustomer(j);
boolean a = nextr.removeCustomer(ci);
boolean b = nextr.removeCustomer(cj);
nextr.addCustomerAt(i,cj);
nextr.addCustomerAt(j,ci);
i++;
j--;
    }
    return nextr;
}
    }
}
```

## C.2 Programs for the VRPTW Formulation

### C.2.1 Data Structures

#### C.2.1.1 Customer.java

```
/*
 * The class customer
 */
public class Customer {
    public double[] coordinates = new double[2];
    public double[] tw = new double[2];
    public double bulk;
    public double arrival;
    // Constructor
    public Customer(double e, double n, double b, double f, double l){
coordinates[0] = e;
coordinates[1] = n;
tw[0] = f;
tw[1] = l;
bulk = b;
    }
    public Customer(double e, double n, double b, ...
    double f, double l, double t){
coordinates[0] = e;
coordinates[1] = n;
tw[0] = f;
tw[1] = l;
bulk = b;
arrival = t;
    }
    public Customer(){}
    /*public Customer copy(Customer c){
double dem = c.demand();
double[] pos = new double[2];
pos = c.position();
double[] tw = new double[2];
tw = c.timewindow();
double time = c.arrivalTime();
Customer newc = new Customer(pos[0], pos[1], dem, tw[0], tw[1], time);
//newc.setDemand(c.demand());
//newc.setPosition(c.position());
//newc.setTW(c.timewindow());
//newc.setTime(c.arrivalTime());
// System.out.println("c" + c.toStringTW());
// System.out.println("newc" + newc.toStringTW());
return newc;
    }*/
 public Customer copy(){
     //double dem = c.demand();
     //double[] pos = new double[2];
```

```
     // pos = c.position();
     // double[] tw = new double[2];
     // tw = c.timewindow();
     // double time = c.arrivalTime();
      Customer newc = new Customer(coordinates[0],coordinates[1], ...
      bulk, tw[0], tw[1], arrival);
//newc.setDemand(c.demand());
//newc.setPosition(c.position());
//newc.setTW(c.timewindow());
//newc.setTime(c.arrivalTime());
// System.out.println("c" + c.toStringTW());
// System.out.println("newc" + newc.toStringTW());
return newc;
    }


    // Method demand, returns the amount of bulk, needed by the given
    // customer
    public double demand(){return bulk;}
    // Method position, returns the utm coordinates for the customer
    public void setDemand(double b){
bulk = b;
    }
    public double[] position(){return coordinates;}
    public void setPosition(double[] coord){
coordinates[0] = coord[0];
coordinates[1] = coord[1];
    }
    // Method timewindow, returns the timewindow for the customer
    public double[] timewindow(){return tw;}
    public void setTW(double[] timew){
tw[0] = timew[0];
tw[1] = timew[1];
    }
    // Method earliest, returns the start of the timewindow
    public double earliest(){return tw[0];}
    // Method latest, returns the end of the timewindow
    public double latest(){return tw[1];}
    // Method setTime, sets the arrivaltime for the vehicle
    public void setTime(double time){
arrival = time;
    }
    // Method arrivalTime returns arrivaltimer for the given customer
    public double arrivalTime(){return arrival;}
    // Method possible, returns 1 if the customer is possible at the
    // given time t.
    public boolean possible(double t){
boolean b = true;
if(t > tw[1]){
    b= false;
}

return b;
    }
```

```
    // toString methods
    public String toString() {
String s = "position (" + coordinates[0] +","+ coordinates[1] + ...
") \t demand " + bulk + "\n";
return s;
    }
    public String toStringTW(double time, Vehicle v) {
double end = time + bulk/(v.effectivity()*v.workinghours());
String s = "pos= (" + coordinates[0] +","+ coordinates[1] + ") ...
\t TW =[" + tw[0] + "," + tw[1] + "], arrivaltime = " + ...
time + ", finnishtime = "+ end +"\n";
return s;
    }
    public String toStringTW(double time) {
String s = "pos= (" + coordinates[0] +","+ coordinates[1] + ") \t ...
TW =[" + tw[0] + "," + tw[1] + "], arrivaltime = " + time + "\n";
return s;
    }
    public String toStringTW(Vehicle v) {
double time = arrival;
double end = time + bulk/(v.effectivity()*v.workinghours());
String s = "pos= (" + coordinates[0] +","+ coordinates[1]...
 + ") \t TW =[" + tw[0] + "," + tw[1] + "], arrivaltime = "...
  + time + ", finnishtime = "+ end +"\n";
return s;
    }
    public String toStringTW() {
String s = "pos= (" + coordinates[0] +","+ coordinates[1] +...
 ") \t TW =[" + tw[0] + "," + tw[1] + "] \n";
return s;
    }

    public String toStringX(){
String s = coordinates[0]+ " \t";
return s;
    }

    public String toStringY(){
String s = coordinates[1] +" \t";
return s;
    }
   public String toStringXY(){
String s = coordinates[0]+ ","+ coordinates[1]+";";
return s;
    }
}
```

### C.2.1.2 Route.java

```
import java.util.*;
/*
```

```
 * The class
 */
public class Route {
    public Vector v = new Vector();

    public Route(){};

    public Route copy(){
Route newr = new Route();
for(int i = 0; i < v.size(); i++){
    Customer cn = (Customer) v.get(i);
    //System.out.println(" cn = " + cn.toStringTW());
    //Customer c = new Customer();
    Customer c = cn.copy();
    //System.out.println(" c = " + c.toStringTW());
    newr.addCustomer(c);
}
return newr;
    }
    // Add customer to the end of the route
    public boolean addCustomer(Customer c){return v.add(c);}
    // Add customer to the specified position of the route
    public void addCustomerAt(int n,Customer c, double[] pF){
v.add(n,c);
for(int i =n+1; i < v.size();i++){
    Customer ci = (Customer) v.get(i);
    double time = pF[i]+ci.arrivalTime();
    ci.setTime(time);
}
    }
    public void addCustomerAt(int n,Customer c, Vehicle k){
double[] pF = pushForward(c, k, n);
v.add(n,c);
for(int i =n+1; i < v.size();i++){
    Customer ci = (Customer) v.get(i);
    double time = pF[i]+ci.arrivalTime();
    ci.setTime(time);
}
    }

    public boolean isRouteFeasible(){
boolean b = true;
for(int i = 0; i<v.size();i++){
    Customer c = (Customer) v.get(i);
    b = (c.arrivalTime()<=c.latest());
    if(!b){
System.out.println("%Customer " + i + " is not feasible");
System.out.println("%"+c.toStringTW(c.arrivalTime()));
break;
    }
}
return b;
    }
    public double setTime(Vehicle car){
double eff = 1/(car.effectivity()*car.workinghours());
```

```
double time = 0;
if(v.size()>0){
    int j = 0;
    Customer cj = (Customer) v.get(j);
    double[] posj = new double[2];
    posj = cj.position();
    double earliestj = cj.earliest();
    time = Math.max(time, earliestj);
    cj.setTime(time);
    time = cj.demand()*eff+time;
    int size = v.size();
    for(int i=1; i < size; i++){
Customer ci = (Customer) v.get(i);
double earliesti = ci.earliest();
double[] posi = new double[2];
posi = ci.position();
double difEast = Math.pow((posj[0]-posi[0]),2);
double difNorth = Math.pow((posj[1]-posi[1]),2);
double distance = Math.sqrt(difEast+difNorth);
time=time+distance/(30*car.workinghours());
time = Math.max(time, earliesti);
ci.setTime(time);
time = time + ci.demand()*eff;
posj = posi;
    }
}
return time;
    }

    // Remove customer from the route, using pushBackward to
    // update arrivaltimes
    public boolean removeCustomer(Customer c, Vehicle veh){
boolean b;
int rem = v.indexOf(c);
int size = v.size();
if(size== 0){
    b = false;
}
else{
    double[] pushback = pushBackward(veh, rem);
    b = v.remove(c);
    if(rem < size-1){
for(int i =rem; i < size-1;i++){
    Customer ci = (Customer) v.get(i);
    double time = ci.arrivalTime()-pushback[i];
    ci.setTime(time);
//  System.out.println("PushBack" + pushback[i]);
}
    }
}
return b;
    }
    // Removing a customer without updating arrivaltime, this is used
    // when removing customers from the dummyroute
    public boolean removeCustomer(Customer c){
```

```
return v.remove(c);
    }
    // Get customer at position i from the route
    public Customer getCustomer(int i){return (Customer) v.get(i);}

    // Get the last customer of the route
    public Customer getLastCustomer()
    {return (Customer) v.lastElement();}
    public boolean containsCustomer(Customer c)
    {return v.contains(c);}

    // The boolean method isCustomerPossible returns 1 if the customer
    // at position i is possible at the time time.
    public boolean isCustomerPossible(int i, double time){
Customer c = (Customer) v.get(i);
boolean b = c.possible(time);
return b;
    }
    public Customer mostUrgent(){
int l = v.size();
Customer curgent = (Customer) v.get(0);
double urgent = curgent.latest();
for(int i=1; i<l; i++){
    Customer c = (Customer) v.get(i);
    double clatest = c.latest();
    if(clatest<urgent){
urgent = clatest;
curgent = c;
    }
}
return curgent;
}

    // Indicates if there are more customers in a route which
    // is possible at the time time
    public boolean moreFeasible(double time){
   int l = v.size();
boolean b = false;
for(int i=0; i<l; i++){
    Customer c = (Customer) v.get(i);
    b = c.possible(time);
    if(b){
break;
    }
}
return b;
    }

    //Number of customers in the route
    public int routeSize(){return v.size();}

    //Remove all customers from the route
    public void empty(){v.clear();}

    //The totalDemand of the Route
```

```
    public double totalDemand(){
      int cost = 0;
      int l = v.size();

      for(int i=0; i<l; i++)
    {
Customer n = (Customer) v.get(i);
cost += n.demand();
    }
return cost;
    }
    // Returns the total distance of the route
    public double totalDistance(){
double cost = 0;
if(v.size()>0){
    int j = 0;
    Customer cj = (Customer) v.get(j);
    double[] posj = new double[2];
    posj = cj.position();
    int size = v.size();
    for(int i=1; i < size; i++){
Customer ci = (Customer) v.get(i);
double[] posi = new double[2];
posi = ci.position();
double difEast = Math.pow((posj[0]-posi[0]),2);
double difNorth = Math.pow((posj[1]-posi[1]),2);
double distance = Math.sqrt(difEast+difNorth);
cost=cost+distance;
posj = posi;
    }
}
return cost;
    }

    //The PushBackward fct.
    //public double[] pushBackward(Route r, Vehicle k, int remove){
    public double[] pushBackward(Vehicle k, int remove){
int length = v.size();
double hoursPerDay = k.workinghours();
double eff = k.effectivity();
double[] pB = new double[length-1];
for(int i = 0; i <remove-1;i++){
    pB[i] = 0;
}
// Calculating PF[start], the difference between old and new arrival
// time.
Customer c1 = (Customer) v.get(remove+1); // The customer just behind
//the customer which should be removed.
double arrivalold = c1.arrivalTime(); // old arrivaltime for this
//customer
//Calculating a new arrivaltime for this customer
Customer cminus1 = (Customer) v.get(remove-1);
double arrivalcminus1 = cminus1.arrivalTime();
double demand = cminus1.demand()/(hoursPerDay*eff);
double dist = findDistance(cminus1,c1)/hoursPerDay;
```

```
double time = arrivalcminus1+demand+dist;
double arrivalnew = Math.max(time,c1.earliest());
pB[remove] = arrivalold - arrivalnew;
for(int i=remove+1;i < length-1;i++){
    Customer ci = (Customer) v.get(i+1);
    double maxPush = Math.max(0,ci.arrivalTime()-ci.earliest());
    pB[i] = Math.min(pB[i-1],maxPush);
}
return pB;
    }
    // Push Forward operation, as described in Solomon pp. 256
    public double[] pushForward(Customer j, Vehicle k, int start){
int length = v.size();
double hoursPerDay = k.workinghours();
double eff = k.effectivity();
double[] pF = new double[length+1];
for(int i = 0; i <start;i++){
    pF[i] = 0;
}
// Calculating PF[start], the difference between old and new arrival
// time.
Customer c1 = (Customer) v.get(start-1);
double arrival = c1.arrivalTime();
double earliest = c1.earliest();
double time = Math.max(arrival,earliest);
//j.setTime(time);
time = time + c1.demand()/(eff*hoursPerDay);
double dist = findDistance(c1,j)/hoursPerDay;
arrival = time+dist;
earliest = j.earliest();
time = Math.max(arrival,earliest);
j.setTime(time);
time = time + j.demand()/(hoursPerDay*eff);
Customer c2 = (Customer) v.get(start);
dist = findDistance(j,c2)/hoursPerDay;
arrival = time+dist;
earliest = c2.earliest();
double newtime = Math.max(arrival,earliest);
double oldtime = Math.max(c2.arrivalTime(),c2.earliest());
pF[start] = newtime - oldtime;
for(int i=start+1;i < length+1;i++){
    Customer ci = (Customer) v.get(i-1);
    double wait = Math.max(0,ci.earliest()-ci.arrivalTime());
    //    System.out.println("wait" + wait);
    pF[i] = Math.max(0,pF[i-1]-wait);
}
return pF;
    }

    public boolean isInsertionTimeFeasible...
    (Customer j, Vehicle k, int start, double[] push){
int size = v.size();
double time = j.arrivalTime(); // arrivaltime is set for customer j
// when calculating the pushForward
double last = j.latest();
```

```
boolean b = (time <= last);
//System.out.println("time < last" + b);
int i = 0;
while(b && i<size){
    Customer ci = (Customer) v.get(i);
    time = ci.arrivalTime();
    last = ci.latest();
    b = (time + push[i] <= last);
    /*if(!b){
System.out.println("b = " +b);
}*/
    i++;
}
return b;
    }

    // Finding distance between two customers in traveltime
    public double findDistance(Customer a, Customer b){
double[] posa = a.position();
double[] posb = b.position();
double difEast = Math.pow((posa[0]-posb[0]),2);
double difNorth = Math.pow((posa[1]-posb[1]),2);
double distance = Math.sqrt(difEast+difNorth);
double velocity = 40;
distance = distance/velocity;
return distance;
    }
    // toString methods
    public String toString(){
String output = new String();

int l = v.size();

for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
  output += c.toString();
      }
  return output;
    }

    public String toStringXY(){
String output = new String();
int l = v.size();
for(int i = 0; i < l; i++){
    Customer c = (Customer) v.get(i);
    output += c.toStringXY();
}
return output;
    }

    public String toStringX(){
String output = new String();

int l = v.size();
```

```
for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
  output += c.toStringX();
      }
  return output;
    }
    public String toStringY(){
String output = new String();

int l = v.size();

for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
  output += c.toStringY();
      }
  return output;
    }
    public String toStringTW(Vehicle veh){
String output = new String();

int l = v.size();

for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
double time = c.arrivalTime();
  output += c.toStringTW(time,veh);
      }
  return output;
    }
    public String toStringTW(){
String output = new String();

int l = v.size();

for(int i=0; i<l; i++)
      {
  Customer c = (Customer) v.get(i);
  output += c.toStringTW();
      }
  return output;
    }
}
```

#### C.2.1.3 Vehicle.java

```
import java.util.*;

public class Vehicle {
```

```
    public double effect;
    public double workhours;
    public Vehicle(double e, double w)
    {
effect = e;
workhours = w;
    };
    //Method effectivity returns the effect of the car
    public double effectivity(){return effect;}
    //Method woorkinghours, returns the workhours per day for the vehicle
    public double workinghours(){return workhours;}
    public String toString(){
String s = "Effect " + effect + "workinghours per day" + workhours;
return s;
    }
}
```

## C.2.2 Solution Methods

### C.2.2.1 FindSolution.java

```
import java.util.*;
/*
 * Find solution, finder initiel lsn. til VRPTW
 * ved brug af time oriented Nearest-Neighbour Heuristic
 */

public class FindSolution{
    public Data data;
    //public Read data;
    public Route[] sol,testsol;
    public Vehicle[] v;

    public FindSolution(String datafile){
System.out.println("clear all, close all");
// udprint til matlabfil
data = new Data(datafile);
//data = new Read(datafile);
int vehicles = 3; // number of vehicles
double eff = 50; // effectivity for vehicle 1 and 2
double eff3 = 35; // effectivity for vehicle 3
double whours = 14; // working hours for vehicle 1 and 2
double whours3 = 10;// working hours for vehicle 3
Vehicle v1 = new Vehicle(eff,whours);
// constructing vehicle 1
Vehicle v2 = new Vehicle(eff,whours);
// constructing vehicle 2
Vehicle v3 = new Vehicle(eff3,whours3);
// constructing vehicle 3
v = new Vehicle[vehicles]; // array with Vehicles
v[0] = v1;
```

```
v[1] = v2;
v[2] = v3;
// Vektor indeholdende alle kunder
Vector customersRaw = data.getData();
// Vektoren laves om til en rute
Route cnorth = new Route();
  for(int i=0;i<customersRaw.size();i++){
    Customer c = (Customer) customersRaw.get(i);
    //System.out.println(c.toStringTW());
    cnorth.addCustomer(c);
}
// Generering af initiel lsn. med nearest neighbor
sol = new Route[4];
/* System.out.println("Tomme ruter");
boolean b1 = sol[0].isRouteFeasible();
System.out.println("%Feasibility of route no 1 = " + b1);
boolean b2 = sol[1].isRouteFeasible();
System.out.println("%Feasibility of route no 2 = " + b2);
boolean b3 = sol[2].isRouteFeasible();
System.out.println("%Feasibility of route no 3 = " + b3);*/
//Initial time
Date startTime = new Date();
// The finishing time
sol = nearestNeighbor(cnorth, v);
Date stopTime = new Date();
long time = stopTime.getTime()-startTime.getTime();
System.out.println("%NNtime = " + time);
/*System.out.println("%Efter Nearest Neighbor");
sol[0].setTime(v1);
boolean b1 = sol[0].isRouteFeasible();
System.out.println("%Feasibility of route no 1 = " + b1);
System.out.println( sol[0].toStringTW(v1));

sol[0].setTime(v1);
b1 = sol[0].isRouteFeasible();
System.out.println...
("%Feasibility of route no 1 efter settime = " + b1);
System.out.println("%Size of rute 1 :" + sol[0].routeSize());
System.out.println( sol[0].toStringTW(v1));
boolean b2 = sol[1].isRouteFeasible();
System.out.println...
("%Feasibility of route no 2 = " + b2);
        sol[1].setTime(v2);
b2 = sol[1].isRouteFeasible();
System.out.println...
("%Feasibility of route no 2 efter settime = " + b2);
System.out.println...
("%Size of rute 2 :" + sol[1].routeSize());
System.out.println( sol[1].toStringTW(v2));

boolean b3 = sol[2].isRouteFeasible();
System.out.println("%Feasibility of route no 3 = " + b3);
sol[2].setTime(v3);
b3 = sol[2].isRouteFeasible();
System.out.println("%Feasibility of route no 3 efter setTime= " + b3);
```

```
System.out.println("%Size of rute 3 :" + sol[2].routeSize());
System.out.println( sol[2].toStringTW(v3));*/


// Check if solution is feasible?
Route inf = sol[3];
int stop = inf.routeSize();
int iteration = 0;
//System.out.println("%Before Insertion heuristic");
//System.out.println("%Number of unrouted Customers" + stop);
Date startTimeins = new Date();
while(stop >0){
    double distExtraDummy = 100000000;
    double distminExtra = distExtraDummy-1;
Customer cmin = inf.getCustomer(0);
    while(distExtraDummy != distminExtra){
distExtraDummy = distminExtra;
//Customer cmin = inf.getCustomer(0);
int startmin = 1;
int currentBest = 0;
Vehicle bestcar = v[0];
for(int r = 0; r < vehicles; r++){
    Vehicle car = v[r];
    Route currentRoute = sol[r];
    int sizeRoute = currentRoute.routeSize();
    for(int s = 1; s < sizeRoute; s++){
for(int i = 0; i < inf.routeSize(); i++){
    Customer c = inf.getCustomer(i);
    double[] pF = currentRoute.pushForward(c, car, s);
    if(currentRoute.isInsertionTimeFeasible(c,car,s,pF)){
  Customer cs1 = currentRoute.getCustomer(s);
Customer cs0 = currentRoute.getCustomer(s-1);
double dist1 = findDistance(cs0,cs1);
double dist2 = findDistance(cs0,c) + findDistance(c,cs1) + ...
c.demand()/(car.effectivity()*car.workinghours());
double distExtra = dist2-dist1;
if(distExtra < distminExtra){
    distminExtra = distExtra;
    cmin = c;
    startmin = s;
    currentBest = r;
    bestcar = car;
}
    }
}
    }
}
// Insert Customer cmin in currentRoute immediately
// before customer nr. s. remove cmin from the route
// inf., update time by using PushForward
// Only when solution becomes time feasible, i.e. if
// (distExtraDummy != distminExtra)
if(inf.routeSize()>0 && distExtraDummy == distminExtra){
    distExtraDummy =distminExtra;
    //System.out.println("%Solution is infeasible");
```

```
    stop = 0;
}
if(distExtraDummy != distminExtra){
    // double[] push = pushForward...
    (sol[currentBest],cmin,bestcar,startmin);
    double[] push = sol[currentBest].pushForward(cmin,bestcar,startmin);
    sol[currentBest].addCustomerAt(startmin, cmin, push);
    inf.removeCustomer(cmin);
    stop--;
    for(int k =0; k<sol[currentBest].routeSize();k++){
//Customer cpush = sol[currentBest].getCustomer(k);
//cpush.setTime(cpush.arrivalTime()+push[k]);
//double arr = cpush.arrivalTime();
distExtraDummy = 100000000;
distminExtra = distExtraDummy-1;
    }
    if(!sol[currentBest].isRouteFeasible()){
System.out.println("%Ups!!!!!!!!!!!!!!!!!!!!!!!!");
    }

    //distExtraDummy = 100000000;
    //distminExtra = distExtraDummy-1;
    iteration++;
}
    }
    stop = inf.routeSize();
if(distExtraDummy == distminExtra){
    stop =0;
}
}
Date stopTimeins = new Date();
long timeins = stopTimeins.getTime()-startTimeins.getTime();
System.out.println("%insertiontime = " + timeins);
//System.out.println("%Efter Insertion heuristic");
System.out.println("%Nu of Unrouted Customers = " +inf.routeSize());
int size = sol.length;
double cost = 0;
double costminusdepot = 0;
for(int i = 0; i < size-1; i++){
    Route rny = sol[i].copy();
         Customer depot = rny.getCustomer(0);
    rny.removeCustomer(depot);
    costminusdepot = costminusdepot + rny.totalDistance();
    cost = cost + sol[i].totalDistance();

}
System.out.println("%initial cost = " + costminusdepot);
//return cost;

/*System.out.println("route0 = [" + ...
sol[0].toStringXY() + "];");
  System.out.println("route1 = [" + ...
  sol[1].toStringXY() + "];");
System.out.println("route2 = [" + ...
sol[2].toStringXY() + "];");
```

```
System.out.println("customer = [" + ...
sol[3].toStringXY() + "];");
System.out.println("plot(route0(:,1),route0(:,2),'r-')");
System.out.println("hold on");
System.out.println("plot(route1(:,1),route1(:,2),'b-')");
System.out.println("hold on");
System.out.println("plot(route2(:,1),route2(:,2),'g-')");
System.out.println("hold on");
System.out.println("plot(customer(:,1),customer(:,2),'m*')");
      System.out.println("figure");
System.out.println("plot(route0(:,1),route0(:,2),'r-')");
System.out.println("hold on");
System.out.println("plot(route1(:,1),route1(:,2),'b-')");
System.out.println("hold on");
System.out.println("plot(route2(:,1),route2(:,2),'g-')");*/
Route rute1 = new Route();
Route rute2 = new Route();
Route rute3 = new Route();
for(int i = 0; i < 10; i++){
    Customer c = sol[0].getCustomer(i);
    rute1.addCustomer(c);
}
for(int i = 0; i < 10; i++){
    Customer c = sol[1].getCustomer(i);
    rute2.addCustomer(c);
}
for(int i = 0; i < 10; i++){
    Customer c = sol[2].getCustomer(i);
    rute3.addCustomer(c);
}
Route testinf = new Route();
/*for(int i = 0; i < 10; i++){
    Customer c = inf.getCustomer(i);
    testinf.addCustomer(c);
    }*/
testsol = new Route[4];
testsol[0]= rute1;
testsol[1]= rute2;
testsol[2]= rute3;
testsol[3]= testinf;
    }
    //getSolution, returns the initial solution
    public Route[] getSolution(){
return sol;
    }
    //getVehicles, returns the vehicles
    public Vehicle[] getVehicles(){
return v;
    }
    public Route[] getTestSolution(){
return testsol;
    }
    //Find distance between 2 customers
    //euklidian distance divided by the velocity
    //which is 40 km/h
```

```
    public double findDistance(Customer a, Customer b){
double[] posa = a.position();
double[] posb = b.position();
double difEast = Math.pow((posa[0]-posb[0]),2);
double difNorth = Math.pow((posa[1]-posb[1]),2);
double distance = Math.sqrt(difEast+difNorth);
double velocity = 30;
distance = distance/velocity;
return distance;
    }

    // Check if two customers are the same
    // benyttes ikke, men skal måske bruges,
    // i forbindelse med kunder
    // der skal besøges to gange
    public boolean identicalCustomer(Customer a, Customer b){
double[] posa = a.position();
double[] posb = b.position();
if(posa[0]==posb[0] && posa[1]==posb[1]){
    return true;
}
else{
    return false;
}
    }

    // Find initial solution in northern Sweden using Solomons
    // Nearest neighbor method.
    private Route[] nearestNeighbor(Route r, Vehicle[] vehicles){
int noOfCars = vehicles.length;
Route[] routes = new Route[noOfCars+1];
int size = r.routeSize();
double time = 0;
int carN = 0;
// Nearest Neighbour procedure is executed, while there are
// more unrouted customers
while(size > 1){
    Vehicle car = vehicles[carN];
    double hoursPerDay = car.workinghours();
    Route newRoute = new Route(); // rute initialiseres
    time=0;
    //Depot er initiel kunde
    double dy = 619800;  // Coordinates for the depot
    double dx = 134200;
    // Omregnet til UTM
    dy = 401.9679740045080;
    dx = 6872.960112292517;
    // demand and timewindows for the depot, dummy values
    double dd = 0;
    double df = 0;
    double dl = 1000000000;
    Customer cmin = new Customer(dy,dx,dd,df,dl); // Depot
    // Depot is first customer on the new Route
    boolean b = newRoute.addCustomer(cmin);
    // Tidsberegning, den er skummel, pga forskel mellem
```

```
    // antallet af arbejdstimer per doegn
    double demandTime = cmin.demand()/...
    (car.effectivity()*hoursPerDay);
    // Her skal indsætte most urgent kunde
    cmin = r.mostUrgent();
    time = cmin.earliest();
    cmin.setTime(time);
    demandTime = cmin.demand()/(car.effectivity()*hoursPerDay);
    time=demandTime+time;
    boolean bool = r.removeCustomer(cmin);
    b = newRoute.addCustomer(cmin);
    size--;
    double el =0;
    int start = 0;
    Customer last = newRoute.getLastCustomer();
    //     time = last.arrivalTime()+last.demand()/...
   (car.effectivity()*hoursPerDay);
    double distmindummy = 100000000;
    double distmin = distmindummy-1;
    while(distmin != distmindummy){
distmin = distmindummy;
last = newRoute.getLastCustomer();
    time = last.arrivalTime()+last.demand()/...
    (car.effectivity()*hoursPerDay);
for(int i=0;i < size;i++){
    Customer l = r.getCustomer(i);
    double arrivalTime =time +findDistance(l, last)/hoursPerDay;
    double first = l.earliest();
    arrivalTime = Math.max(first,arrivalTime);
    //     if(r.isCustomerPossible(i,arrivalTime)){
    if(arrivalTime < l.latest()){
double dist = costTW(newRoute,cmin,car,el);
if(dist < distmin){
    distmin = dist;
    cmin = l;
}
    }
}
// The customer which is nearest to the last customer in the
// route, are deleted from the route containing nonrouted
// customers and added to the current route.
double arrivalTime =time+findDistance(cmin, last)/hoursPerDay;
if(arrivalTime> cmin.latest()){
    System.out.println("last " + last.toStringTW(car));
    System.out.println("distance " + ...
    findDistance(cmin, last)/hoursPerDay);
    System.out.println("time " + time);
    System.out.println("cmin"+ cmin.toStringTW(arrivalTime));
}
bool = r.removeCustomer(cmin);
b = newRoute.addCustomer(cmin);
//cmin.setTime(arrivalTime+time);
arrivalTime = Math.max(cmin.earliest(),arrivalTime);
cmin.setTime(arrivalTime);
size--;
```

```
demandTime = cmin.demand()/(car.effectivity()*hoursPerDay);
time=demandTime+arrivalTime;
int lastCustomer = newRoute.routeSize()-1;
int sizer = r.routeSize();
int y = lastCustomer+1;
int yu = sizer+y;
    }
    routes[carN] = newRoute;
    carN++;
    if(carN >noOfCars-1){
routes[carN] = r;
Route re = routes[carN];
// String strin = re.toString();
//System.out.println(strin);
int jl = r.routeSize();
int rsize = r.routeSize();
for(int i=0;i<rsize;i++){
    Customer cust = r.getCustomer(i);
    String rest = cust.toStringXY();
}
size=0;
    }
}
// Try adding customers to exsisting route ...
using an isertion heuristic
// Skal evt. laves til en selvst. fkt.
Route rout = routes[0];
String str=routes[0].toString();
double siz = routes[0].routeSize();
// System.out.println("size" + siz + str);
return routes;
    }

    // The earliest possible time, customer c can be visited
    // Kan evt. optimeres, hvis sluttiden for sidste kunde i ruten
    // medsendes
    public double earliestPosTime(Route r,...
     Customer c, Vehicle k, double elast){
double[] tw = c.timewindow();
double ac = tw[0];
int size = r.routeSize();
double time = 0;
//     double slast = 0;
    //     double t = 0;

if(size!=0){
    Route rnew = new Route();
    //rute der ikke indeholder slutknude
    for(int i=0; i < size-1; i++){
Customer cus =  r.getCustomer(i);
boolean b = rnew.addCustomer(cus);
    }
    Customer lastCustomer = r.getCustomer(size-1);
    double workhours = k.workinghours();
    // Rekursiv beregning af elast
```

```
    //elast = earliestPosTime(rnew, lastCustomer, k);
    double slast = lastCustomer.demand()/k.effectivity();
    double t = findDistance(c, lastCustomer);
    // Besvaerlig tidsudregning
    time = slast+t/workhours;
}

double e = Math.max(ac, elast + time);
return e;
    }
    // The temporal distance between the last customer in the route
    // and a new customer, eik, is the earliest possible time for the
    // last customer in the route r
    public double temporalDistance...
    (Route r, Customer j, Vehicle k,double eik){
int size = r.routeSize();
Customer i = r.getCustomer(size-1);
// double ejk = 0;
double ejk = earliestPosTime(r,j,k,eik);
Route rnew = new Route();
for(int l=0; l < size-1; l++){
    Customer cus =  r.getCustomer(l);
    boolean b = rnew.addCustomer(cus);
    }
// double eik = 0;
// double eik = earliestPosTime(rnew,i,k);
double sik = k.effectivity()*i.demand();
double tijk = ejk - (eik + sik);
return tijk;
    }
    // The urgency of serving customer i after the customer
    // in the route r. - defined by solomon
    public double urgency(Route r, Customer j, Vehicle k){
int size = r.routeSize();
Customer i = r.getCustomer(size-1);
double[] twi = i.timewindow();
double[] twj = j.timewindow();
double bj = twi[1];
double ai = twj[0];
double sik = k.effectivity()*i.demand();
double tij = findDistance(i,j);
double vijk = bj - (ai + sik + tij);
return vijk;
    }

    // Calculates the cost of appending Customer j to Route r
    // it is calculated with respect to urgency
    //and a "temporal distance"
    // defined by Solomon
    public double costTW(Route r, Customer j, ...
    Vehicle k,double elast){
double a = 1;
double b = 0;
double c = 0;
int size = r.routeSize();
```

```
Customer i = r.getCustomer(size-1);
double dij = findDistance(i, j);
double tijk = temporalDistance(r, j, k,elast);
double vijk = urgency(r, j, k);
double cost = a * dij + b * tijk + c * vijk;
return cost;
    }
    /*    // Push Forward operation, as described in Solomon pp. 256
    public double[] pushForward(Route r, ...
    Customer j, Vehicle k, int start){
int length = r.routeSize();
double hoursPerDay = k.workinghours();
double eff = k.effectivity();
double[] pF = new double[length+1];
for(int i = 0; i <start-1;i++){
    pF[i] = 0;
}
// Calculating PF[start], the difference between old and new arrival
// time.
Customer c1 = r.getCustomer(start-1);
double arrival = c1.arrivalTime();
double earliest = c1.earliest();
double time = Math.max(arrival,earliest);
j.setTime(time);
time = time + c1.demand()/(eff*hoursPerDay);
double dist = findDistance(c1,j)/hoursPerDay;
arrival = time+dist;
earliest = j.earliest();
time = Math.max(arrival,earliest);
time = time + j.demand()/(hoursPerDay*eff);
Customer c2 = r.getCustomer(start);
dist = findDistance(j,c2)/hoursPerDay;
arrival = time+dist;
earliest = c2.earliest();
double newtime = Math.max(arrival,earliest);
double oldtime = Math.max(c2.arrivalTime(),c2.earliest());
pF[start-1] = newtime - oldtime;
for(int i=start;i < length+1;i++){
    Customer ci = r.getCustomer(i-1);
    double wait = Math.max(0,ci.earliest()-ci.arrivalTime());
    pF[i] = Math.max(0,pF[i-1]-wait);
}
return pF;
}*/
    // Push Forward operation, analogue to the
    //push forward operation
    /*    public double[] pushBackward(Route r, Vehicle k, int remove){
int length = r.routeSize();
double hoursPerDay = k.workinghours();
double eff = k.effectivity();
double[] pB = new double[length-1];
for(int i = 0; i <remove-1;i++){
    pB[i] = 0;
}
// Calculating PF[start], the difference
```

```
//between old and new arrival
// time.
Customer c1 = r.getCustomer(remove+1);
// The customer just behind
//the customer which should be removed.
double arrivalold = c1.arrivalTime(); // old arrivaltime for this
//customer
//Calculating a new arrivaltime for this customer
Customer cminus1 = r.getCustomer(remove-1);
double arrivalcminus1 = cminus1.arrivalTime();
double demand = cminus1.demand()/(hoursPerDay*eff);
double dist = findDistance(cminus1,c1)/hoursPerDay;
double time = arrivalcminus1+demand+dist;
double arrivalnew = Math.max(time,c1.earliest());
pB[remove] = arrivalold - arrivalnew;
for(int i=remove+1;i < length-1;i++){
    Customer ci = r.getCustomer(i+1);
    double maxPush = Math.max(0,ci.arrivalTime()-ci.earliest());
    pB[i] = Math.max(0,pB[i-1]-maxPush);
}
return pB;
}*/
    // Using lemma 1.1 from solomon, pp. 256. the neccesary and
    // sufficient condition for time feasibility
    /*    public boolean isInsertionTimeFeasible(Route r, ...
    Customer j, Vehicle k, int start){
int size = r.routeSize();
double[] push = r.pushForward(j, k, start);
double time = j.arrivalTime(); // ikke sikkert j har arrivaltime
//fra pushForward, ellers skal arrivaltime beregnes som i
//pushforward
double last = j.latest();
boolean b = (time < last);
int i = 0;
while(b && i<size){
    Customer ci = r.getCustomer(i);
    time = ci.arrivalTime();
    last = ci.latest();
    b = (time + push[i] <= last);

    i++;
}
return b;
}*/
}
```

### C.2.2.2 SimulatedAnnealing.java

```
import java.util.*;
/*
 * SimulatedAnnealing, using the metaheuristic...
  simulated annealing to
```

```
 * improve an initial solution
 */

public class SimulatedAnnealing{
    FindSolution sol = new FindSolution("dataTW.txt");
    public SimulatedAnnealing(){
//System.out.println("clear all, close all");
//Route[] initialsol = sol.getSolution();
Route[] initialsol = sol.getSolution();
int noOfRoutes = initialsol.length;
// System.out.println("no Of Routes" + noOfRoutes);
Route inf = new Route();
Route[] solution = new Route[3];
    solution[0] = initialsol[0];
    solution[1] = initialsol[1];
    solution[2] = initialsol[2];
if(noOfRoutes > 3){
    inf = initialsol[3];
}

System.out.println("initialroute0 = ...
[" + solution[0].toStringXY() + "];");
System.out.println("initialroute1 =...
 [" + solution[1].toStringXY() + "];");
System.out.println("initialroute2 =...
 [" + solution[2].toStringXY() + "];");
System.out.println("initialunrouted =...
 [" + inf.toStringXY() + "];");
System.out.println("%No. ofunrouted customers" + inf.routeSize());
      System.out.println("figure");
System.out.println("plot(initialroute0(:,1),...
initialroute0(:,2),'r-')");
System.out.println("hold on");
System.out.println("plot(initialroute1(:,1),...
initialroute1(:,2),'b-')");
System.out.println("hold on");
System.out.println("plot(initialroute2(:,1),...
initialroute2(:,2),'g-')");
Vehicle[] cars = sol.getVehicles();
int noOfCars = cars.length;
boolean b1 = solution[0].isRouteFeasible();
System.out.println("%Feasibility of route no 1 = " + b1);
boolean b2 = solution[1].isRouteFeasible();
System.out.println("%Feasibility of route no 2 = " + b2);
boolean b3 = solution[2].isRouteFeasible();
System.out.println("%Feasibility of route no 3 = " + b3);
//for(double t = 1; t < 35; t++){
    SA(solution,cars,15,0.98,50);
    //}
    }
    // The simulated annealing procedure
    // sol: the solution, without eventually unscheduled customers
    // t: the start temperature
    // r: the reduction factor
    // l: number of iterations between the temperature is reduced
```

```
    public void SA(Route[] sol, Vehicle[] cars, ...
    double t, double r, int l){
Vector costv = new Vector();
Vector costbestv = new Vector();
boolean stopkriterie = false;
Route[] oldsol = new Route[3];
Route r1 = sol[0].copy();
Route r2 = sol[1].copy();
Route r3 = sol[2].copy();
oldsol[0]= r1;
oldsol[1]= r2;
oldsol[2]= r3;
int infeasible = 0;
int nonaccept = 0;
int accept = 0;
int improving = 0;
double acceptstep = 0;
double nonacceptstep = 0;
double costold = costDistance(oldsol); //Distance
//double costold = costTime(oldsol,cars); //Time
double costtime = costTime(oldsol,cars); //Time
System.out.println("%initially cost time" + costtime);
double currentbest = costold;
String beststr = new String();
beststr = "best = [";
String coststr = new String();
coststr = "cost = [";
//costv.addElement(costold);
//costbestv.addElement(currentbest);
beststr += currentbest;
coststr += costold;
System.out.println("%Cost-start =" + costold);
int ite = 0;
while(!stopkriterie){
    //Initial time
    Date startTime = new Date();
    for(int i = 0; i < l; i++){
sol[0].empty();
sol[1].empty();
sol[2].empty();
sol[0] = oldsol[0].copy();
sol[1] = oldsol[1].copy();
sol[2] = oldsol[2].copy();
// find ny lsn. i Neighboorhood
// dvs. find ny lsn. vha. customerChangeRoute
int ins = (int) Math.floor(3*Math.random());
int rem = (int) Math.floor(3*Math.random());
while(ins == rem){
    ins = (int) Math.floor(3*Math.random());
rem = (int) Math.floor(3*Math.random());
}
int cust = (int) Math.ceil((sol[rem].routeSize()-1)*...
Math.random());
Customer c = sol[rem].getCustomer(cust);
```

```
double bestrsize = sol[ins].totalDistance()+...
sol[rem].totalDistance()+700;
boolean timef = false;
int bestinsert = sol[ins].routeSize()+10;
for(int insert = 1; insert <sol[ins].routeSize() ;insert++){
    Route[] candidatesol = new Route[3];
    Route cr1 = new Route();
    Route cr2 = new Route();
    Route cr3 = new Route();
    cr1 = sol[0].copy();
    cr2 = sol[1].copy();
    cr3 = sol[2].copy();
    candidatesol[0]= cr1;
    candidatesol[1]= cr2;
    candidatesol[2]= cr3;
    Customer ccand = candidatesol[rem].getCustomer(cust);
    double[] pf = new double[candidatesol[ins].routeSize()+1];
    pf = candidatesol[ins].pushForward(ccand, cars[ins], insert);
    if(candidatesol[ins].isInsertionTimeFeasible...
    (ccand, cars[ins], insert, pf))
{
    if(cust < candidatesol[rem].routeSize()-1){
    candidatesol[rem].removeCustomer(ccand,cars[rem]);
    }
    else{
candidatesol[rem].removeCustomer(ccand);
    }
    double currentrsize = candidatesol[ins].totalDistance() ...
    + candidatesol[rem].totalDistance();
        if(currentrsize < bestrsize && ...
        candidatesol[ins].isRouteFeasible()){
       bestinsert = insert;
       bestrsize = currentrsize;
       timef = true;
   }
}
    //NYT HERFRA, optimering af kode:-)
    else{
Customer cinsert = candidatesol[ins].getCustomer(insert-1);
double cearliest = cinsert.earliest();
if(cearliest > ccand.latest()){
    //System.out.println("%Rest of Route infeasible");
    insert = sol[ins].routeSize();
}
    }
}
if(timef){
sol[ins].addCustomerAt(bestinsert, c, cars[ins]);
sol[rem].removeCustomer(c);
}
double candidatecost = costDistance(sol); //Distance
//double candidatecost = costTime(sol,cars); //Tid
// The finishing time
Date stopTime = new Date();
long time = stopTime.getTime()-startTime.getTime();
```

```
//System.out.println("%Iteration time = " + time);
if(candidatecost < costold){
    costold = candidatecost;
    currentbest = Math.min(costold,currentbest);
    //System.out.println("%better sol, new cost = "+ costold);
    improving++;
    oldsol[0].empty();
    oldsol[1].empty();
    oldsol[2].empty();
    oldsol[0] = sol[0].copy();
    oldsol[1] = sol[1].copy();
    oldsol[2] = sol[2].copy();
  beststr +=  "," + currentbest;
coststr += "," + costold;
    //costv.add(costold);
    //costbestv.add(currentbest);
ite++;
//   System.out.println("%ite =" + ite);
}
if(candidatecost>costold){
    double prob = Math.exp((costold-candidatecost)/t);
    nonaccept++;
    nonacceptstep++;
    ite++;
    //System.out.println("%ite =" + ite);
    if(Math.random()<prob){
costold = candidatecost;
accept++;
//System.out.println("%worse sol chosen, ...
new cost = " + costold);
acceptstep++;
oldsol[0].empty();
oldsol[1].empty();
oldsol[2].empty();
oldsol[0] = sol[0].copy();
oldsol[1] = sol[1].copy();
oldsol[2] = sol[2].copy();
beststr +=  "," + currentbest;
coststr += "," + costold;
//costv.add(costold);
//costbestv.add(currentbest);
    }
}
    }
    t = r*t;
    //nonacceptstep = nonacceptstep - acceptstep;
    //double ratio = acceptstep/nonacceptstep;
    //System.out.println("ratio[" + t + "] = ...
    " + ratio + ";");
    //nonacceptstep = 0;
    //acceptstep = 0;
    //System.out.println("%t = " + t);
    //System.out.println("%ite = " + ite);
    ite++;
    if(ite >= 7000){
```

```
    double ratio = acceptstep/nonacceptstep;
    //System.out.println("%nonacceptstep = "...
     + nonacceptstep);
    //System.out.println("%acceptstep = " + acceptstep);
    //System.out.println("%acceptance ratio = " + ratio);
stopkriterie = true;
//double candidatecost = costDistance(sol);
//double oldcost = costDistance(oldsol);
double candidatecost = costTime(sol,cars);
double oldcost = costTime(oldsol,cars);
System.out.println("%improve" + improving);
System.out.println("%accept" + accept);
nonaccept = nonaccept - accept;
System.out.println("%nonaccept" + nonaccept);
infeasible = ite - (improving+ nonaccept+accept);
System.out.println("%infeasible" + infeasible);
System.out.println("%new candidatecost = "...
 + candidatecost);
System.out.println("%new cost = " + oldcost);
        System.out.println("finalroute0 = ...
        [" + oldsol[0].toStringXY() + "];");
System.out.println("finalroute1 = ...
[" + oldsol[1].toStringXY() + "];");
System.out.println("finalroute2 = ...
[" + oldsol[2].toStringXY() + "];");
System.out.println("figure");
System.out.println("plot(finalroute0(:,1),...
finalroute0(:,2),'r-')");
System.out.println("hold on");
System.out.println("plot(finalroute1(:,1),...
finalroute1(:,2),'b-')");
System.out.println("hold on");
System.out.println("plot(finalroute2(:,1),...
finalroute2(:,2),'g-')");
oldsol[0].setTime(cars[0]);
oldsol[1].setTime(cars[1]);
oldsol[2].setTime(cars[2]);
System.out.println("%Feasibility of route 1" + ...
oldsol[0].isRouteFeasible());
System.out.println("%Feasibility of route 2" + ...
 oldsol[1].isRouteFeasible());
System.out.println("%Feasibility of route 3" + ...
oldsol[2].isRouteFeasible());
//Udskrift til Matlab filer - currentbest og cost fkt.
coststr += "];";
beststr += "];";
System.out.println(beststr);
System.out.println(coststr);
System.out.println("sb = size(best);");
System.out.println("sc = size(cost);");
System.out.println("b1 = linspace(0,sb(2),sb(2));");
System.out.println("c1 = linspace(0,sc(2),sc(2));");
System.out.println("figure");
System.out.println("plot(b1,best,'r-')");
System.out.println("hold on");
```

```
System.out.println("plot(c1,cost,'b-')");
    }
}
    }

                /***********************
                *                      *
                *    COST FUNCTIONS    *
                *                      *
                ***********************/
    // used in the SA-procedure to evaluate A given solution.

    // Method to calculate how infeasible the solution is
    // i.e., how many customer is not scheduled.
    public double costfct(Route[] sol, Route dummy){
int costinf = costInfeasibility(sol, dummy);
double cost = costDistance(sol, dummy);
cost = cost + costinf*100000;
return cost;
    }
    public int costInfeasibility(Route[] sol, Route dummy){
int cost = dummy.routeSize();
return cost;
    }
    // Method to calculate the total amount of time used.
    public double costTime(Route[] sol, Route dummy, Vehicle[] cars){
int size = sol.length;
double cost = 0;
for(int i = 0; i < size; i++){
    Customer last = sol[i].getCustomer((sol[i].routeSize()-1));
    double effectivity = (cars[i].effectivity()*...
    cars[i].workinghours());
    cost = cost + last.arrivalTime() + last.demand()/effectivity;
}
cost = cost*24;
return cost;
    }
    // Method to calculate the total amount of time used.
    public double costTime(Route[] sol, Vehicle[] cars){
double cost = 0;
for(int i = 0; i < sol.length; i++){
    cost = cost + sol[i].totalDistance()/...
    (30*cars[i].workinghours());
    double eff = 1/(cars[i].effectivity()*...
    cars[i].workinghours());
    cost = cost + sol[i].totalDemand()*eff;
}
cost = cost*24;
return cost;
    }
    // Method to calculate the total distance in the solution
    public double costDistance(Route[] sol, Route dummy){
int size = sol.length;
double cost = 0;
double costminusdepot = 0;
```

```
for(int i = 0; i < size; i++){
    Route rny = sol[i].copy();
    Customer depot = rny.getCustomer(0);
    rny.removeCustomer(depot);
    costminusdepot = costminusdepot + rny.totalDistance();
    cost = cost + sol[i].totalDistance();

}
//return cost;
return costminusdepot;

    }

    public double costDistance(Route[] sol){
int size = sol.length;
double cost = 0;
double costminusdepot = 0;
for(int i = 0; i < size; i++){
    Route rny = sol[i].copy();
    Customer depot = rny.getCustomer(0);
    rny.removeCustomer(depot);
    costminusdepot = costminusdepot + ...
    rny.totalDistance();
    cost = cost + sol[i].totalDistance();

}
//return cost;
return costminusdepot;

    }
}
```

### C.2.2.3 SteepestDescent.java

```
import java.util.*;
/*
 * SimulatedAnnealing, using the metaheuristic
 * simulated annealing to
 * improve an initial solution
 */

public class SteepestDescent2{
    FindSolution sol = new FindSolution("dataTW.txt");
    public SteepestDescent2(){
//System.out.println("clear all, close all");
//Route[] initialsol = sol.getSolution();
Route[] initialsol = sol.getSolution();
int noOfRoutes = initialsol.length;
// System.out.println("no Of Routes" + noOfRoutes);
Route inf = new Route();
Route[] solution = new Route[3];
    solution[0] = initialsol[0];
```

```
    solution[1] = initialsol[1];
    solution[2] = initialsol[2];
if(noOfRoutes > 3){
    inf = initialsol[3];
}

System.out.println("initialroute0 = ...
[" + solution[0].toStringXY() + "];");
System.out.println("initialroute1 = ...
[" + solution[1].toStringXY() + "];");
System.out.println("initialroute2 = ...
[" + solution[2].toStringXY() + "];");
System.out.println("initialunrouted = ...
[" + inf.toStringXY() + "];");
System.out.println("%No. ofunrouted customers" + inf.routeSize());
      System.out.println("figure");
System.out.println("plot(initialroute0(:,1),...
initialroute0(:,2),'r-')");
System.out.println("hold on");
System.out.println("plot(initialroute1(:,1),...
initialroute1(:,2),'b-')");
System.out.println("hold on");
System.out.println("plot(initialroute2(:,1),...
initialroute2(:,2),'g-')");
Vehicle[] cars = sol.getVehicles();
int noOfCars = cars.length;
boolean b1 = solution[0].isRouteFeasible();
System.out.println("%Feasibility of route no 1 = " + b1);
boolean b2 = solution[1].isRouteFeasible();
System.out.println("%Feasibility of route no 2 = " + b2);
boolean b3 = solution[2].isRouteFeasible();
System.out.println("%Feasibility of route no 3 = " + b3);
StD(solution,cars);
    }
    // The simulated annealing procedure
    // sol: the solution, without eventually ...
    //unscheduled customers
    // t: the start temperature
    // r: the reduction factor
    // l: number of iterations between the temperature is reduced

    public void StD(Route[] sol, Vehicle[] cars){
Vector costv = new Vector();
Vector costbestv = new Vector();
boolean stopkriterie = false;
Route[] oldsol = new Route[3];
Route r1 = sol[0].copy();
Route r2 = sol[1].copy();
Route r3 = sol[2].copy();
oldsol[0]= r1;
oldsol[1]= r2;
oldsol[2]= r3;
double costold = costDistance(oldsol);
double currentbest = costold;
String beststr = new String();
```

```
beststr = "best = [";
beststr += currentbest;
System.out.println("%Cost-start =" + costold);
int ite = 0;
while(!stopkriterie){
    stopkriterie = true;
    //for(int i = 0; i < l; i++){
    sol[0].empty();
    sol[1].empty();
    sol[2].empty();
    sol[0] = oldsol[0].copy();
    sol[1] = oldsol[1].copy();
    sol[2] = oldsol[2].copy();
    // Investigate solutions in the neighborhood
    int bestins = 0;
    int bestrem = 0;
    int bestcust = 0;
    int bestinsert = sol[0].routeSize()+100;
    double bestrsize = costDistance(oldsol);
    //Initial time
    Date startTime = new Date();
    for(int ins = 0; ins < 3; ins++){
for(int rem = 0; rem < 3; rem++){
    if(ins !=rem){
//System.out.println("ins != rem");
for(int cust =1; cust <sol[rem].routeSize();cust++){
    Customer c = sol[rem].getCustomer(cust);
    //double bestrsize = sol[0].totalDistance() + ...
    sol[1].totalDistance()+sol[2].totalDistance();
    //stopkriterie = true;
    for(int insert = 1; insert <sol[ins].routeSize()...
     ;insert++){
// System.out.println("insert" + insert);
Route[] candidatesol = new Route[3];
Route cr1 = new Route();
Route cr2 = new Route();
Route cr3 = new Route();
cr1 = sol[0].copy();
cr2 = sol[1].copy();
cr3 = sol[2].copy();
candidatesol[0]= cr1;
candidatesol[1]= cr2;
candidatesol[2]= cr3;
Customer ccand = candidatesol[rem].getCustomer(cust);
double[] pf = new double[candidatesol[ins].routeSize()+1];
pf = candidatesol[ins].pushForward(ccand, cars[ins], insert);
if(candidatesol[ins].isInsertionTimeFeasible...
(ccand, cars[ins], insert, pf))
    {
candidatesol[ins].addCustomerAt(insert,ccand,cars[ins]);
if(cust < candidatesol[rem].routeSize()-1){
    //stopkriterie = false;
    candidatesol[rem].removeCustomer(ccand,cars[rem]);
}
else{
```

```
    candidatesol[rem].removeCustomer(ccand);
}

double currentrsize = costDistance(candidatesol);
if(currentrsize < bestrsize && ...
candidatesol[ins].isRouteFeasible()){
    stopkriterie = false;
    bestins = ins;
    bestrem = rem;
    bestcust = cust;
    bestinsert = insert;
    bestrsize = currentrsize;
    //timef = true;
}
    }
//NYT HERFRA, optimering af kode:-)
else{
    Customer cinsert = candidatesol[ins].getCustomer(insert-1);
    double carrival = cinsert.arrivalTime();
    if(carrival > ccand.latest()){
//System.out.println("%Rest of Route infeasible");
insert = sol[ins].routeSize();
    }
}
    }
}
}
}
    }
    // The finishing time
    Date stopTime = new Date();
    long time = stopTime.getTime()-startTime.getTime();
    System.out.println("%Iteration time" + time);
    if(!stopkriterie){
Customer bestCustomer = sol[bestrem].getCustomer(bestcust);
sol[bestins].addCustomerAt(bestinsert, ...
bestCustomer, cars[bestins]);
sol[bestrem].removeCustomer(bestCustomer);
    }
    double candidatecost = costDistance(sol);
    costold = candidatecost;
    beststr += "," + costold;
    //currentbest = Math.min(costold,currentbest);
    System.out.println("%better sol, new cost = "+ costold);
    oldsol[0].empty();
    oldsol[1].empty();
    oldsol[2].empty();
    oldsol[0] = sol[0].copy();
    oldsol[1] = sol[1].copy();
    oldsol[2] = sol[2].copy();
    //costv.add(costold);
    //costbestv.add(currentbest);
    //if(stopkriterie){
}
oldsol[0].setTime(cars[0]);
```

```
oldsol[1].setTime(cars[1]);
oldsol[2].setTime(cars[2]);
// Check om ruter er feasible!!!
System.out.println("%Feasibility of route 1" +...
 oldsol[0].isRouteFeasible());
System.out.println("%Feasibility of route 2" +...
 oldsol[1].isRouteFeasible());
System.out.println("%Feasibility of route 3" +...
 oldsol[2].isRouteFeasible());
// Udskriv ruter til Matlab
System.out.println("optroute0 = [" ...
+ oldsol[0].toStringXY() + "];");
System.out.println("optroute1 = [" ...
+ oldsol[1].toStringXY() + "];");
System.out.println("optroute2 = [" ...
+ oldsol[2].toStringXY() + "];");
      System.out.println("figure");
System.out.println("plot(optroute0(:,1),...
optroute0(:,2),'r-')");
System.out.println("hold on");
System.out.println("plot(optroute1(:,1),...
optroute1(:,2),'b-')");
System.out.println("hold on");
System.out.println("plot(optroute2(:,1),...
optroute2(:,2),'g-')");
// Plot cost fkt.
System.out.println(beststr + "];");
System.out.println("sb = size(best);");
System.out.println("b1 = linspace(0,sb(2),sb(2));");
System.out.println("figure");
System.out.println("plot(b1,best,'r-')");
    }


                /************************
 *                      *
 *    COST FUNCTIONS    *
 *                      *
 ************************/
    // used in the SA-procedure to evaluate A given solution.

    // Method to calculate how infeasible the solution is
    // i.e., how many customer is not scheduled.
    public double costfct(Route[] sol, Route dummy){
int costinf = costInfeasibility(sol, dummy);
double cost = costDistance(sol, dummy);
cost = cost + costinf*100000;
return cost;
    }
    public int costInfeasibility(Route[] sol, Route dummy){
int cost = dummy.routeSize();
return cost;
    }
    // Method to calculate the total amount of time used.
    public double costTime(Route[] sol, ...
```

```
    Route dummy, Vehicle[] cars){
int size = sol.length;
double cost = 0;
for(int i = 0; i < size; i++){
    Customer last = sol[i].getCustomer((sol[i].routeSize()-1));
    double effectivity = (cars[i].effectivity()...
    *cars[i].workinghours());
    cost = cost + last.arrivalTime() + ...
    last.demand()/effectivity;
}
return cost;
    }
    // Method to calculate the total distance in the solution
    public double costDistance(Route[] sol, Route dummy){
int size = sol.length;
double cost = 0;
double costminusdepot = 0;
for(int i = 0; i < size; i++){
    Route rny = sol[i].copy();
    Customer depot = rny.getCustomer(0);
    rny.removeCustomer(depot);
    costminusdepot = costminusdepot + rny.totalDistance();
    cost = cost + sol[i].totalDistance();

}
//return cost;
return costminusdepot;

    }

    public double costDistance(Route[] sol){
int size = sol.length;
double cost = 0;
double costminusdepot = 0;
for(int i = 0; i < size; i++){
    Route rny = sol[i].copy();
    Customer depot = rny.getCustomer(0);
    rny.removeCustomer(depot);
    costminusdepot = costminusdepot + rny.totalDistance();
    cost = cost + sol[i].totalDistance();

}
//return cost;
return costminusdepot;

    }
}
```

## C.3 Read Procedure and MainClass

#### C.3.0.4 Data.java

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

// The data object contains the data for the problem.
class Data
{
    private StreamTokenizer st;
    public Vector v = new Vector();
    public Data(String filename){
readAndParseFile(filename);
    }
    public Vector getData(){
return v;
    }
    // Method readAndParseFile, reads the text file, which contains data
    private void readAndParseFile(String filename){
     openFile(filename);
int i = 0;
while(st.ttype != StreamTokenizer.TT_EOF){
    if(st.ttype == StreamTokenizer.TT_WORD){
boolean flaga = next();
    }
    if(st.ttype == StreamTokenizer.TT_NUMBER){
// Position in longitude and latitude
double n =   st.nval;
// Laves til degrees
n = 0.0001*n;
boolean flagb2 = next();
double e = st.nval;
e = 0.0001*e;
boolean flagd2 = next();
// Position in UTM - (skal bruges for at kunne beregne
// afstande i meter)
double[] coord = geoToUtm(n,e);
// positionerne skaleres op, saa enhederne for afstande
// bliver km i stedet for m.
coord[0]=coord[0]/1000;
coord[1]=coord[1]/1000;
// Demand - kundens demand
double d = st.nval;
boolean flagc1 = next();
// Dato for levering iflg. gammel plan,
// benyttes til beregning af TW
double week = st.nval;
boolean flaga3 = next();
double day = st.nval;
boolean flaga8 = next();
// Vehicle - which vehicles is used to
```

```
// service this customer in Daniscos sol
double vehicle = st.nval;
boolean flaga4 = next();
// TimeWindow - beregnet efter gammel dlv. date
day = week*7 + day;
double f = (day-5);
double l = (day+5);
Customer c = new Customer(coord[0],coord[1],d,f,l);
v.add(c);
    i++;
    }
    else{
boolean flagelse = next();
    }
}
    }
/*
 * Get the next number from the open file
 */
    private boolean next()
    {
boolean flag = true;
try{ st.nextToken(); }
catch(IOException e)
    {
System.err.println("Error: Failed to get next number. ...
Is the file open?");
System.exit(-1);
flag = false ;
    }
return flag;
    }

    // Omregning fra lattitude and longitude to UTM coordinates,
    // Den originale matlab kode er implementeret af
    // Anna BO Jensen,
    private double[] geoToUtm(double LatDeg, double LonDeg){

        double a = 6378137.0; // Ellipsoidal semimajor axis, GRS80
        double f = 1 / 298.257223563;  // Ellipsoidal flattening, GRS80
        double b = a*(1 - f);         // Ellipsoidal semiminor axis
        double e2 =  f*(2.0 - f);      // Ellipsoidal second eccentricity
        double n = (a - b) / (a + b);  // Third flattening
        double e4 = Math.pow(e2,2);    // Second eccentricity squared

        // UTM zone 32 or 33
        double FE = 500000;            // False Easting
        double FN = 0.0;               // False Northing
        double cm = 15;                 // Central meridian, zone 33
// double LatDeg =latitude[i];
//double LonDeg =longitude[i];
// Transformation variables
double DeltaLonDeg = LonDeg - cm;   // Longitude difference to
//central meridian
double Lat = LatDeg * Math.PI/180;      // Convert to radians
```

```
double Lon = LonDeg * Math.PI/180;
double DeltaLon = DeltaLonDeg * Math.PI/180;
double coslat = Math.cos(Lat);
double sinlat = Math.sin(Lat);
double tanlat = Math.tan(Lat);
double N = a / (Math.sqrt(1 - (e2*Math.pow(sinlat,2))) );  //
//Radius of curvature in prime vertical
double eta2 = (e2*Math.pow(coslat,2)) / (1 - e2);
// Meridian arc length, dm, equations from Poder and Engsager, 1998
double m0 = 1.0;
double Qm = (m0*(a / (1 + n))) * (1 + Math.pow(n,2)/4 + Math.pow(n,4)/64);
double p2 = -3*n/2 + 9*Math.pow(n,3)/16;
double p4 = 15*Math.pow(n,2)/16 - 15*Math.pow(n,4)/32;
double p6 = -35*Math.pow(n,3)/48;
double p8 = 315*Math.pow(n,4)/512;
double Sum = (p2*Math.sin(2*Lat)) + (p4*Math.sin(4*Lat))+...
(p6*Math.sin(6*Lat)) + (p8*Math.sin(8*Lat));
double dm = Qm * (Lat + Sum);
//dmK = 6208563.012            % Control value for lat=56.0 and
//    long=10.0 from KMSTRans
// UTM conversion, equations from Pearson, 1990
double fx = 1 - Math.pow(tanlat,2) + eta2;
double fy = 5 - Math.pow(tanlat,2) + 9*eta2;

double X = N *( (DeltaLon*coslat) +
(((Math.pow(DeltaLon,3))*(Math.pow(coslat,3))*fx) / 6) );
double Y = dm +  N*( (sinlat*coslat*Math.pow(DeltaLon,2)/2)+(sinlat*...
(Math.pow(coslat,3))*(Math.pow(DeltaLon,4))*fy/24) );
double E = 0.9996*X + FE;
N = 0.9996*Y + FN;
double[] output = new double[2];
output[0]=E;
output[1]=N;
return output;
   }
    // Metode til at åbne fil
    private void openFile(String filename){
try{ st = new StreamTokenizer(new FileReader(filename)); }
catch(FileNotFoundException e)
    {
System.err.println("File \"" + filename + "\" not found");
System.exit(0);
    }
    }

}
```

#### C.3.0.5 MainClass.java

```
import java.util.*;
import java.util.Date;
/*
 * The main class.
 */
class MainClass
{
     public static void main(String args[])
    {
// SimulatedAnnealing SA = new SimulatedAnnealing();
SteepestDescent2 StD = new SteepestDescent2();
//FindSolution sol = new FindSolution("dataTW.txt");
//NyFindSolution sol = new NyFindSolution("dataTW.txt");
    }
}
```

## C.4 Generating Data

#### C.4.0.6 Parametertuning.java

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

// The data object contains the data for the problem.
class Parametertuning
{
    private StreamTokenizer st;
    public Vector v = new Vector();
    public Parametertuning(String filename){
readAndParseFile(filename);
    Vector original = getData();

    //System.out.println("hej");
    //System.out.println("originalsize" + original.size());
    //Vector redData = reduceData(original, 40);
    }
    public Vector getData(){
return v;
    }
    // Method readAndParseFile, reads the text
    //file, which contains data
    private void readAndParseFile(String filename){
     openFile(filename);
int i = 0;
while(st.ttype != StreamTokenizer.TT_EOF){
    if(st.ttype == StreamTokenizer.TT_WORD){
boolean flaga = next();
```

```
    }
    if(st.ttype == StreamTokenizer.TT_NUMBER){
// Position in longitude and latitude
double n =   st.nval;
// Laves til degrees
n = 0.0001*n;
boolean flagb2 = next();
double e = st.nval;
e = 0.0001*e;
boolean flagd2 = next();
// Position in UTM - (skal bruges for at kunne beregne
// afstande i meter)
double[] coord = geoToUtm(n,e);
// positionerne skaleres op, saa enhederne for afstande
// bliver km i stedet for m.
coord[0]=coord[0]/1000;
coord[1]=coord[1]/1000;
// Demand - kundens demand
double d = st.nval;
boolean flagc1 = next();
// Dato for levering iflg. gammel plan,
// benyttes til beregning af TW
double week = st.nval;
boolean flaga3 = next();
double day = st.nval;
boolean flaga8 = next();
// Vehicle - which vehicles is used to
// service this customer in Daniscos sol
double vehicle = st.nval;
boolean flaga4 = next();
// TimeWindow - beregnet efter gammel dlv. date
day = week*7 + day;
double f = (day-5);
double l = (day+5);
Customer c = new Customer(coord[0],coord[1],d,f,l);
v.add(c);
    i++;
    }
    else{
boolean flagelse = next();
    }
}
    }
/*
 * Get the next number from the open file
 */
    private boolean next()
    {
boolean flag = true;
try{ st.nextToken(); }
catch(IOException e)
    {
System.err.println("Error: Failed to get next number. ...
Is the file open?");
System.exit(-1);
```

```
flag = false ;
    }
return flag;
    }

    // Omregning fra lattitude and longitude to UTM coordinates,
    // Den originale matlab kode er implementeret af
    // Anna BO Jensen,
    private double[] geoToUtm(double LatDeg, double LonDeg){

        double a = 6378137.0; // Ellipsoidal semimajor axis, GRS80
        double f = 1 / 298.257223563;  // Ellipsoidal flattening, GRS80
        double b = a*(1 - f);         // Ellipsoidal semiminor axis
        double e2 =  f*(2.0 - f);      // Ellipsoidal second eccentricity
        double n = (a - b) / (a + b);  // Third flattening
        double e4 = Math.pow(e2,2);    // Second eccentricity squared

        // UTM zone 32 or 33
        double FE = 500000;            // False Easting
        double FN = 0.0;               // False Northing
        double cm = 15;                 // Central meridian, zone 33
// double LatDeg =latitude[i];
//double LonDeg =longitude[i];
// Transformation variables
double DeltaLonDeg = LonDeg - cm;   // Longitude difference to
//central meridian
double Lat = LatDeg * Math.PI/180;      // Convert to radians
double Lon = LonDeg * Math.PI/180;
double DeltaLon = DeltaLonDeg * Math.PI/180;
double coslat = Math.cos(Lat);
double sinlat = Math.sin(Lat);
double tanlat = Math.tan(Lat);
double N = a / (Math.sqrt(1 - (e2*Math.pow(sinlat,2))) );  //
//Radius of curvature in prime vertical
double eta2 = (e2*Math.pow(coslat,2)) / (1 - e2);
// Meridian arc length, dm, equations from Poder and Engsager, 1998
double m0 = 1.0;
double Qm = (m0*(a / (1 + n))) * (1 + Math.pow(n,2)/4 + Math.pow(n,4)/64);
double p2 = -3*n/2 + 9*Math.pow(n,3)/16;
double p4 = 15*Math.pow(n,2)/16 - 15*Math.pow(n,4)/32;
double p6 = -35*Math.pow(n,3)/48;
double p8 = 315*Math.pow(n,4)/512;
double Sum = (p2*Math.sin(2*Lat)) + (p4*Math.sin(4*Lat))...
+(p6*Math.sin(6*Lat)) + (p8*Math.sin(8*Lat));
double dm = Qm * (Lat + Sum);
//dmK = 6208563.012            % Control value for lat=56.0 and
//    long=10.0 from KMSTRans
// UTM conversion, equations from Pearson, 1990
double fx = 1 - Math.pow(tanlat,2) + eta2;
double fy = 5 - Math.pow(tanlat,2) + 9*eta2;

double X = N *( (DeltaLon*coslat) +
(((Math.pow(DeltaLon,3))*(Math.pow(coslat,3))*fx) / 6) );
double Y = dm +  N*( (sinlat*coslat*Math.pow(DeltaLon,2)/2)+(sinlat*...
(Math.pow(coslat,3))*(Math.pow(DeltaLon,4))*fy/24) );
```

```
double E = 0.9996*X + FE;
N = 0.9996*Y + FN;
double[] output = new double[2];
output[0]=E;
output[1]=N;
return output;
   }
    // Metode til at åbne fil
    private void openFile(String filename){
try{ st = new StreamTokenizer(new FileReader(filename)); }
catch(FileNotFoundException e)
    {
System.err.println("File \"" + filename + "\" not found");
System.exit(0);
    }
    }
    public Vector reduceData(Vector org, int nsize){
Vector reduced = new Vector();
//int orgsize = org.size();
while(org.size() > nsize){
    //System.out.println("orgsize" + orgsize)
    int n = (int) Math.floor(Math.random()*(org.size()));
    org.remove(n);

}
/*for(int i=0; i<nsize;i++){
    Customer c = (Customer) reduced.get(i);
    String s = c.toString();
    System.out.println(s);
    }*/
return org;

    }
    public Vector changeDemand(Vector org){
Vector change = new Vector();
for(int i = 0; i <org.size();i++){
    Customer c = (Customer) org.get(i);
    Customer cnew = c.copy();
    change.add(cnew);
}
    int j = 0;
while(org.size()>0){
    int n = (int) Math.floor(Math.random()*(org.size()));
    Customer cold = (Customer) org.get(n);
    org.remove(cold);
    double dem = cold.demand();
    Customer cj = (Customer) change.get(j);
    cj.setDemand(dem);
    //System.out.println("demand" +  dem);
    j++;
}
return change;
    }
    public String toString(Vector data){
String s = "east\tnorth\tdemand\tearliest\tlatest\n";
```

```
for(int i = 0; i <data.size();i++){
    Customer c = (Customer) data.get(i);
    s = s + c.toString();
}
return s;
    }

}
```

# Bibliography


[1] The EU Commissioner for Agriculture and Rural Development. Eu radically reforms its sugar sector to give producers long-term competitive future. *Eurpean Commission, Press Releases*, 2005.

[2] Alf Duch-Pedersen. Danisco fremtidssikrer sukkeraktiviteterne. *Danisco, Press Releases*, 2006.

[3] Paolo Toth and Vigo Danielle. The vehicle routing problem. 2002.

[4] Laurence A. Wolsey. *Integer Programming.* Wiley, 1998. ISBN: 0-471-28366-5.

[5] Niklas Kohl, Jacques Desrosiers, Oli B. G. Madsen, Marius M. Solomon, and Francois Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[6] M. W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 1985.

[7] Gilbert Laporte and Frédéric Semet. Classical heuristic for the vehicle routing problem. *Les Cahiers du GERAD*, 1998.

[8] Michel Gendreau, Gilbert Laporte, and Jean-Yves Potvin. Metaheuristic for the vehicle routing problem. *Les Cahiers du GERAD*, 1998.

[9] Alex Van Breedam. Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, 86:p. 480–490, 1993.

[10] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41, 1993.

[11] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *University of Bologna*, 1998.

[12] Olli Bräysy and Michel Gendrau. Vehicle routing problem with timewindows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):p. 104–118, 2005.

[13] Olli Bräysy and Michel Gendrau. Vehicle routing problem with timewindows, part ii: Metaheuristics. *Transportation Science*, 39(1):p. 119–139, 2005.

[14] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):p. 254–265, 1987.

[15] J. Y. Potvin and J. M. Rousseau. A parallel routebuilding algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66:p. 331–340, 1993.

[16] K. C. Tan, L. H. Lee, Q. L. Zhu, and K. Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15, 2001.

[17] J. Czech Zbigniew. Parallel simulated annealing for the vehicle routing probem with time windows. *Proceedings 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 2002.

[18] Martin Descrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):p. 342–354, 1992.

[19] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The travelling salesman problem. *Handbools in OR & MS*, 7:p. 225–330, 1995.

[20] Finn Bo Madsen and Anna B. O. Jensen. Nye koordinatsystemer i danmark. 1999.

[21] Anna B. O. Jensen. Geotoutm (matlab file).

[22] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw-Hill Book Company, 2000.

[23] Per S. Laursen. *Generelle Optimeringsheuristikker - en introduktion*. DIKU, 1994.

[24] Tim Duncan. Experiments in the use of neighborhood search techniques for vehicle routing. 1995.