# Solving a Vehicle Routing Problem with cross docking

Málfríður Guðmundsdóttir
Anna Margrét Pétursdóttir

LYNGBY
MAY 2005

IMM

Trykt af IMM, DTU

# Preface

This is a Master's thesis carried out at the section for Operations Research (OR), Informatics and Mathematical Modelling (IMM), Technical University of Denmark (DTU). The thesis is based on 7 months work, accounting 35 ETCS units.

The project was proposed by Jakob Birkedal Nielsen at Transvision and the supervisor was Jesper Larsen.

Lyngby, May 2005

Málfríður Guðmundsdóttir
s010347

Anna Margrét Pétursdóttir
s031211

# Abstract

This thesis is about the solution of a pickup and delivery problem with a central cross docking station. All orders must first be picked up at their various locations and moved to the cross docking station. Then the orders can be delivered to their destinations.

The fleet of vehicles is considered to be homogeneous and each vehicle has a limited capacity. Each order has a specific size, along with exact pickup location and destination.

An extensive mathematical model of the problem is presented and implemented in GAMS. The LP relaxation of the problem is then solved in order to get a lower bound. The problem is then solved by using Simulated Annealing (SA). Two versions of the SA algorithm are implemented and then improved, but they differ in the way they find a neighborhood solution. These two versions are then compared. The quality of the solutions from the better algorithm is then considered by looking at the evolution of the objective values with time, when running the algorithms for a longer period of time. Furthermore, the importance of a good initial solution is investigated for both algorithms.

**Keywords**: Pickup and delivery, cross docking station, routing, VRP, SA.

# Contents

# Chapter 1

# Introduction

This project is originally proposed by Transvision but has been slightly modified by us. In this chapter we will describe the problem, the data we have got to work with and how we are going to solve the problem.

## 1.1 Verbal description of the problem

The information we got from Transvision regarding the problem have been gathered here in the following problem description.

The problem is a pickup and delivery problem, where all orders are supposed to go through a central cross docking station.

A cross docking station is a temporary storage where orders come in and are sorted. An illustration of a cross docking station can be seen in figure 1.1. In this case the orders come in from being picked up and are then sorted onto the vehicles delivering them. At the end of a work day the cross docking station should be empty.

In this problem, pickup and delivery routes are to be constructed with the following arrangement in mind. The orders should first be picked up and moved to the cross docking station. Then they can either be unloaded at the cross docking station or they can be left on the vehicle, if that same vehicle is driving the delivery route that order is delivered on. The orders that have been unloaded at the cross docking station are then loaded onto other vehicles which then deliver them to their destinations. Loading an unloading orders obviously takes some time and includes some expenses.

The vehicle fleet is homogeneous and the same vehicles are used for both pickup and delivery. Each vehicle has a maximum working time. The vehicles should not be left unused for too long at the cross docking station.

The problem can be made more complicated, for example by putting time windows on the customers. That would be done in the way that each customer could only be visited within a certain time limit, for example only before noon or after certain time of the day. We can also complicate the problem by minimizing the number of vehicles used. To do that we could take the routes and assign them to certain vehicles, as a result the vehicles

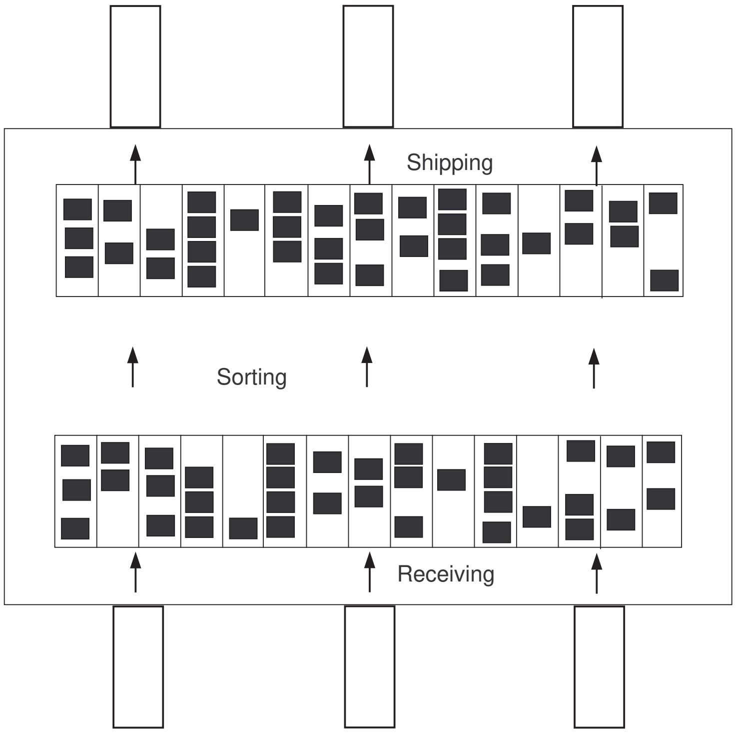


Figure 1.1: This figure shows a cross docking station operates.

would then probably be driving more than one route. We could then penalize the number of vehicles used in.

The objective of this problem is to minimize the total driving time of the vehicles along with the number of vehicles required.

## 1.2 The data

Data for the problem was provided by Transvision. The data is not from an actual problem, but has been constructed with real locations in mind. These locations can be seen in figure 1.2, the locations are all in Denmark.

The data consists of approximately 1100 orders that are divided onto 5 days. Each day can be considered as a separate problem.

The coordinates of the locations where the orders should be picked up at and delivered to are presented in meters. Furthermore, the distances used are assumed to be euclidian. In addition to the coordinates, the size of each order is given in number of pallets.

Standard data for the vehicles is also given, it includes the coordinates of the cross docking station where the vehicles are supposed to begin and end each route. In addition to the coordinates we have the capacity and the average speed of the vehicles, along with the time window in which the vehicles are available, it is the same for all vehicles. In principle there are infinitely many vehicles available and they are all identical.

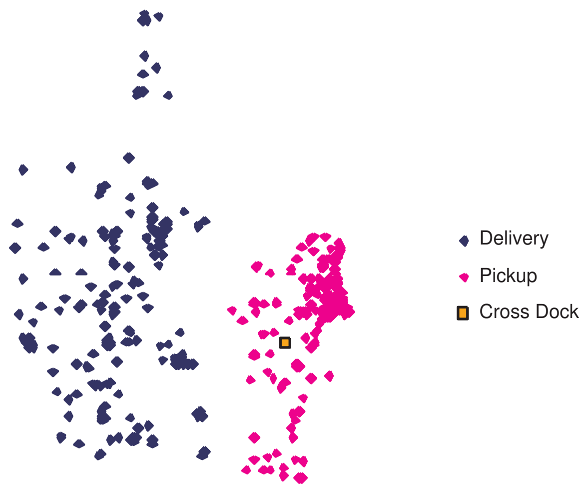


Figure 1.2: Figure showing the locations of the orders.

Calculation of time spent on loading and unloading the vehicles is identical for the pickup or delivery of each order. It consists of a constant time for each stop plus time proportional to the number of pallets loaded or unloaded.

## 1.3 Thesis overview

In chapter 2 we analyze the problem. We discuss the Vehicle Routing Problem (VRP) and how we can use it when making a mathematical model of our problem.

The mathematical problem is then presented in chapter 3. Every constraint is defined and explained along with the variables and constants used.

In chapter 4 the solution of the mathematical model is described. The implementation of the model in GAMS and how the mathematical model was improved in the process, is discussed. Furthermore, we illustrate how we were able to solve a small problem to optimality using our GAMS code. Then we explain why solving our full size problem to optimality is not an option in this case.

In chapter 5 we state our choice of metaheuristic and the chosen one, simulated annealing, is described. Moreover, we explain how we implemented two versions of the simulated annealing algorithm, both in words and pseudo codes. More implementation details follow.

The improvements of our algorithms are discussed in chapter 6. We introduce a Local Search Algorithm which we use to improve the structure of the routes. In addition, we introduce a method to try to eliminate routes that only transport a single order.

In chapter 7 we describe the parameter tuning. We present all the parameters involved and explain the procedure of tuning. The data sets used and the result of the parameter tuning are also discussed.

The testing of the algorithm is described in chapter 8 as well as the results. The results are illustrated in tables and graphs.

In chapter 9 the conclusion is presented, and finally in chapter 10 we discuss our of further work.

Note that the terms cross docking station and depot will be used equally when talking about our problem in this report. They both denote the same thing, namely what up until now has been referred to as the cross docking station. Furthermore, the words vehicles and trucks are used equally.

# Chapter 2

# Analysis of the problem

In this chapter we will discuss how we, based on the verbal description of the problem, derive the mathematical model. This problem exhibits many similarities to the Vehicle Routing Problem (VRP) and we are going to take advantage of that in the formulation of the problem. In this chapter we will also describe the VRP and how it is modelled. At last we will discuss how we use the VRP model as a basis for our model.

## 2.1 Our problem

The first step in the solution of the problem is to make a mathematical model of the problem. In order to make that step it has to be clear what our assumptions are and what we want to get out of the process or what kind of solution we are looking for.

We have identified the basic requirements of a model for our problem. There are in particular four main things to consider. Those are the following:

1. The total driving time is to be minimized as well as the number of vehicles used.
2. We need some information about the routes, which orders should be picked up on each route and which routes are handled by which vehicle.
3. The vehicles have a maximum capacity and all the vehicles used are identical.
4. The orders have to arrive at the cross docking station before they can be delivered.

Since the locations of pickup and delivery are in this case in different regions, as was shown in figure 1.2 in chapter 1, we have decided to assume we have two different fleets of vehicles. More accurately one for pickup and another one for delivery.

Obviously there are many more details that need to be considered in the model, but these are the main requirements and assumptions we make to the basic version of a model for this problem.

## 2.2 The Vehicle Routing Problem

The problem we are dealing with resembles the Vehicle Routing Problem (VRP) also denoted the Capacitated Vehicle Routing Problem (CVRP). We will call it VRP.

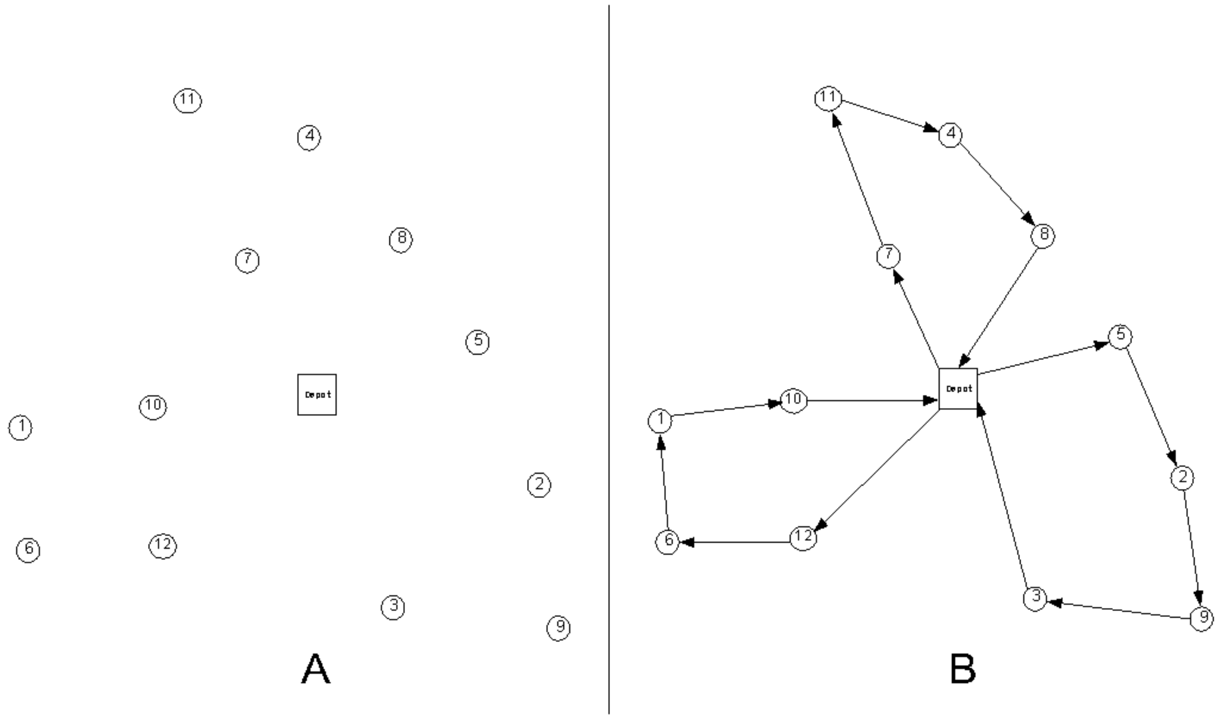


Figure 2.1: An example of a solution of a VRP, (A) shows the unsolved problem with the customers and the depot, and (B) shows the routes in the solution found.

### 2.2.1 Introduction to the VRP

The VRP was originally introduced by Dantzig and Ramser in 1959 [7] and has been widely studied since. It is a complex combinatorial optimization problem.

A basic VRP can be described as follows. Given a fleet of homogeneous vehicles with a given capacity, a common depot, and several costumer demands, find the set of routes with overall minimum route cost which will service all the demands. Every route starts and ends at the depot and the routes must be configured in such a way that each costumer is visited only once and just by one of the routes.

Figure 2.1 shows a typical VRP. (A) is the unsolved problem, while (B) shows a solution of the problem. The numbered circles represent the customers and the box is the depot. In part (B) of the figure where the routes have been constructed, the arrows show the configuration of the routes. As can be seen, this particular problem has 12 customers and its solution consists of 3 different routes.

### 2.2.2 Formulation of the VRP

Here we formulate a version of the VRP model taken from the book "The Vehicle Routing Problem" by Toth and Vigo [7]. This model is the so-called three-index vehicle flow formulation of the VRP.

Having a three-indexed model allows us to indicate which vehicle traverses an arc $(i, j)$, or which vehicle drives from customer $i$ to customer $j$. This allows for more involved constraints to be imposed on the routes than if we only have a two-indexed model.

In this model, the variable $x_{i,j,k}$ is binary and shows whether arc $(i,j)$ is traversed by vehicle $k$ in the optimal solution or not. The variable $y_{i,k}$ takes value 1 if customer $i$ is served by vehicle $k$ in the optimal solution and takes value 0 otherwise. Here, $V$ is the set of customers and 0 represents the depot. The three-index model is given in the following.

$$\min \sum_{i \in V} \sum_{j \in V} c_{i,j} \sum_{k=1}^{K} x_{i,j,k} \tag{2.1}$$

Subject to:

$$\sum_{k=1}^{K} y_{i,k} = 1 \qquad \forall\ i \in V \backslash \{0\}, \tag{2.2}$$

$$\sum_{k=1}^{K} y_{0,k} = K, \tag{2.3}$$

$$\sum_{j \in V} x_{i,j,k} = \sum_{j \in V} x_{j,i,k} = y_{i,k} \qquad \forall\ i \in V, k = 1, ..., K, \tag{2.4}$$

$$\sum_{i \in V} d_i y_{i,k} \leq C \qquad \forall\ k = 1, ..., K, \tag{2.5}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{i,j,k} \geq y_{h,k} \qquad \forall\ S \subseteq V \backslash \{0\}, h \in S,\ k = 1, ..., K, \tag{2.6}$$

$$y_{i,k} \in \{0,1\} \qquad \forall\ i \in V, k = 1, ..., K, \tag{2.7}$$

$$x_{i,j,k} \in \{0,1\} \qquad \forall\ i,j \in V, k = 1, ..., K. \tag{2.8}$$

The first constraints 2.2 impose that each customer is visited exactly once. Constraints 2.3 make sure that $K$ vehicles leave the depot and constraints 2.4 impose that the same vehicle enters and leaves the given customer. Then there are capacity constraints in 2.5, where $C$ denotes the capacity of the vehicles and $d_i$ the size of order $i$. Constraints 2.6 impose the connectivity of the route performed by vehicle $k$. Those constraints may be replaced by subtour elimination constraints. Finally 2.7 and 2.8 show that $x_{i,j,k}$ and $y_{i,k}$ are binary variables.

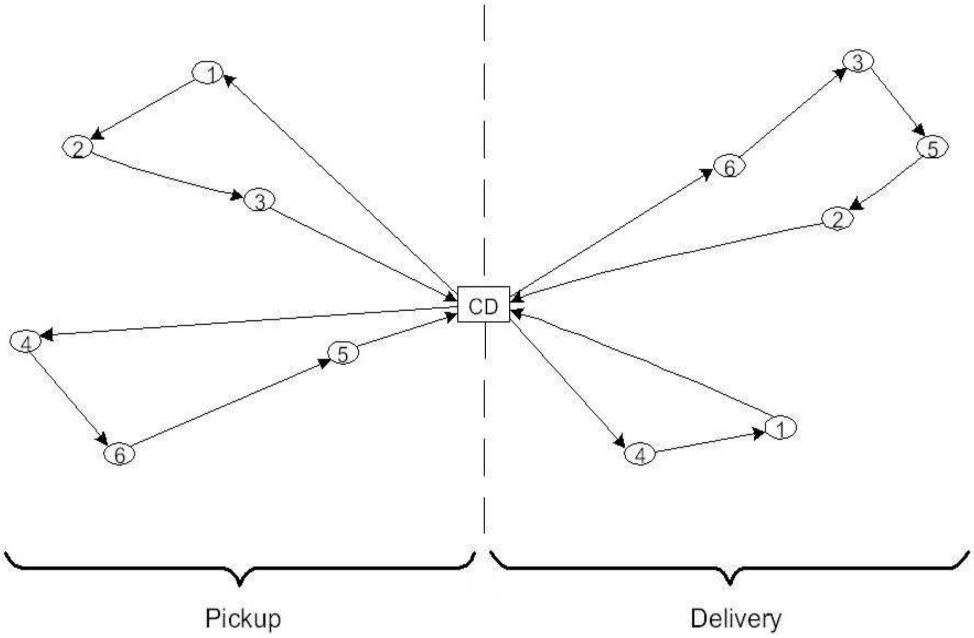


Figure 2.2: An example of a solution of a small instance of our problem.

### 2.2.3 The formulation of our problem

After studying the VRP we found it to be a good foundation to build our model on. We choose to view the problem as two VRP's, coupled together by the cross docking station. The two problems are the pickup part and the delivery part of the problem.

Figure 2.2 shows how a solution to our problem looks like for a small problem with only 6 orders. Looking at this figure it can be seen how the problem can be divided up in two VRP problems, one for pickup and another for delivery.

Reviewing the requirements we stated in section 2.1 for a model of our problem, we can see that the first three are almost all subjects that the VRP deals with. These can up to some degree be considered separately for the pickup and delivery parts of the problem.

However, the pickup and delivery parts of our problem are dependent on each other in the way that each order is first picked up and then delivered. This implies, as stated in the fourth requirement we made in section 2.1, that the orders will always have to have been picked up and moved to the cross docking station before they can be delivered.

This, along with other details of our problem calls for some additional constraints not found in the VRP model. Nevertheless, the VRP is a good basis for our model since it can be used to solve the pickup and delivery problems individually. How we combine the VRP's for the two problems and develop the mathematical model further will be described in the next chapter.

## 2.3 Literature review

In this section we will discuss various solution methods for the VRP. Information about the methods was gathered from the book *The Vehicle Routing Problem* edited by P. Toth and D. Vigo [7].

**About the Capacitated vehicle routing problem**

The Capacitated vehicle routing problem CVRP is a generalization of the Travelling Salesman Problem TSP. The TSP, which is a well known problem, is NP-hard[1] which makes the CVRP also a NP-hard problem.

It is difficult to solve NP-hard problems and very time consuming. Because of this, no algorithm has been developed that can solve a large instance of a VRP to optimality. However, there exist a few exact methods which have mainly been used to solve the VRP to optimality, but that is only for small instances of the problem. These are for example, branch-and-bound and branch-and-cut.

**Solving with exact methods**

The branch-and-bound method has been used in a great extent in recent decades to solve the VRP and the many variations of the problem. The method is mostly successful for solving instances involving very few customers. One of the most successful branch-and-bound algorithms, proposed by Fisher [2], has been able to solve instances up to 100 customers.

The branch-and-cut method has been successfully used on the TSP problem when finding optimal solutions for large instances. However, not much has been published about applying the branch-and-cut to the CVRP but this topic is being researched more.

**Solving with classical heuristics**

The classical heuristics were mostly developed between 1960 and 1990. The classical heuristics can be classified into three groups. Constructive heuristics, two-phase heuristics and improvement methods.

These methods find good solutions in a reasonable time but they only perform a limited search in the solution space. However, there is another class of heuristics that has been applied on the VRP, namely metaheuristics. Metaheuristics have been growing for the last decade. Unlike classical heuristics, metaheuristics search the solution space thoroughly. They typically use neighborhood search rules, rearrange the solutions or memory structures and the quality of the solutions they obtain is much higher than the ones the classical heuristics produce. The solutions produced by metaheuristics acquire longer running time than the solutions obtained by the classical heuristics.

**Solving with metaheuristics**

There is a clear difference between metaheuristics and the classical heuristics when it comes to accepting new solutions during the solution process, because the metaheuristics allow

[1] Non-deterministic Polynomial-time Hard

infeasible solutions and accept a solution worse than the incumbent one during the process. There are six types of metaheuristics that we know of which have been applied to the VRP problem. They are the Simulated Annealing (SA), Deterministic Annealing (DA), Tabu Search (TS), Genetic Algorithms (GS), Ant Systems (AS) and Neural Networks (NN).

The first three algorithms start with an initial solution and then find a neighborhood solution they can possibly accept as a new solution. New solutions could be accepted even though their objective value is lower than the value of the incumbent solution. This is done until a certain stopping criterion is met. GA use population of solutions and use the best elements to populate new ones. AS uses information gathered from previous iterations to construct new solutions. NN is a learning procedure where a set of weights is gradually adjusted until it finds an acceptable solution.

The method that is considered to be the most effective approach and give over all best results when applied on the VRP is TS. Neural networks and operations based on GA are out performed. Procedures based on SA, DA and AS give good solutions but are not quite competitive to TS. The data these conclusions are made from are not very large, i.e. up to 200 customers, so testing the metaheuristics with larger data sets could show that they perform differently.

As has been said before, metaheuristics produce much better solutions than classical heuristics but instead metaheuristics are very time consuming. Good metaheuristic implementations often produce solution values which are less than 0.5% from the optimal or best known value, whereas classical methods typically gives values between 2% and 10%. The classical methods produce their solutions within seconds since they build their solutions by following simple rules, whereas with metaheuristics it can take hours and the procedure of finding a solution is much more complicated.

Future work will be to try to get as good solutions as the metaheuristics give but in less time than it takes with the algorithms used today. This will most likely be achieved by speeding up the running time of the metaheuristics rather than focusing on the classical methods.

## 2.4 Our approach

After the analysis of the problem, described in the chapter, we have concluded how we are going to approach this problem.

In order to solve this problem we have chosen to start by making a mathematical model of the problem. Since the problem is NP-hard and therefore a large scale optimization problem we find it appropriate to use metaheuristics to solve it. Because we have no real life data to compare the results from our metaheuristic with, we are going to solve the mathematical model with LP relaxation in order to get a lower bound.

# Chapter 3

# The mathematical model of the problem

Our problem is significantly more complex than the Vehicle Routing Problem (VRP) and in fact our problem can be viewed as two VRP models linked together by the depot/cross-docking station as described in chapter 2. Besides, we have additional conditions which are not taken into account in the basic VRP model in section 2.2.2. The VRP can although be used as a starting point as there are many similarities between our problem and the basic VRP.

In this chapter the model is defined and explained in detail. The whole model can also be found in Appendix A, where all parameters and variables used are explained and every constraint involved is listed.

## 3.1 The parameters and the basic variables of the model

To be able to define the objective function of the model we need to introduce the parameters and the basic variables of the model. The rest of the variables will be defined and explained as they come along.

The parameters of this model are like in the basic VRP model, $i$ and $j$ which are used for customers or more accurately the locations of orders. The total number of orders to be transported is $N$ and their locations are numbered from 1 to $N$. We also have an order $N+1$ added to the $N$ orders and let it, along with $i = 0$, represent the depot. So 0 represents the out-depot or that is the depot when driving out and $N+1$ represents the in-depot, namely the depot when returning to it.

Furthermore, there are two indices for routes, $k$ for pickup routes and $l$ for delivery routes. Those parameters do not represent the vehicles like in the VRP above because in this case we want each vehicle to be able to drive more than one route in the optimal solution. The total number of pickup/delivery routes possible is $K/L$ respectively. These will be explained more thoroughly in section 3.3.3.

The binary variables $x_{i,j,k}$ and $z_{i,j,l}$ define the pickup and delivery routes with the minimum driving time. In this problem each arc between $i$ and $j$ should never be travelled more than once. Therefore they are binary and get the value 1 if order $j$ is picked up or delivered directly after order $i$ on route $k$ or $l$ respectively, and the value 0 otherwise.

## 3.2 The objective function

The objective of this model is to minimize the driving time of the vehicles. Since the problem consists of two VRP's linked together by the depot, the objective function here is similar to the one in the VRP model. The difference is that the objective function from the VRP model is double, that is, we are minimizing the driving time for the pickup as well as the delivery routes and here they can be viewed as separate VRP's.

In the data given for the problem, the coordinates for the locations of pickup and delivery for each order $i$ are given. Since the distances can be considered euclidian, it is possible to calculate the time it takes to travel between all the locations where orders are to be picked up, along with the travelling time between each location and the cross docking station. The same applies to the locations to be delivered to. This gives us the matrices $A_{i,j}$ and $B_{i,j}$ for pickups and deliveries respectively. They hold the driving times between locations of orders $i$ and $j$ and are used when minimizing the driving time involved.

Given the information above we can now define the objective function for this problem, namely:

$$\min \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} \left( A_{i,j} \sum_{k=1}^{K} x_{i,j,k} + B_{i,j} \sum_{l=1}^{L} z_{i,j,l} \right) + TotSerTime \tag{3.1}$$

In addition to the driving time of the vehicles, the pickup and delivery of the orders in this problem involve some service time. The $TotSerTime$ or the total service time for loading and unloading the vehicles is included in the objective function since it is a part of the total working time. The service time at the pickup and delivery locations is constant for any given instance, but the service time at the cross docking station is variable depending on the number of routes driven. Equation 3.21 shown along with the time constraints here below, shows how the $TotSerTime$ is calculated.

## 3.3 The Constraints

Similar to the VRP model we have constraints to make sure we get routes that visit each and every customer once and only once. In addition to the two binary variables introduced before we use the binary variables $y_{i,k}$ and $v_{i,l}$ to indicate whether order $i$ is picked up or delivered on route $k$ or $l$ respectively, or not. Furthermore, we say that order $i$ is picked up on route $k/l$ if the vehicle on that route leaves the location of the given order.

$$\sum_{j=1}^{N} x_{i,j,k} = y_{i,k}, \quad \forall\ i = 1, \ldots, N,\ k = 1, \ldots, K \tag{3.2}$$

$$\sum_{j=1}^{N} z_{i,j,l} = v_{i,l}, \quad \forall\ i = 1, \ldots, N,\ l = 1, \ldots, L \tag{3.3}$$

We guaranty that every order is picked up and delivered by adding the following constraints.

$$\sum_{k=1}^{K} y_{i,k} = 1, \quad \forall\ i = 1, \ldots, N \tag{3.4}$$

$$\sum_{l=1}^{L} v_{i,l} = 1, \quad \forall\ i = 1, \ldots, N \tag{3.5}$$

These also make sure each order is only picked up and delivered on one route.

### 3.3.1 Flow constraints

It is also important that the same vehicle enters and leaves the location of the given order, moreover, if we arrive at a given location we have to leave it again. Therefore we need to introduce the flow conservation constraints [7] to make sure the routes are formed correctly.

$$\sum_{j=0}^{N+1} x_{i,j,k} = \sum_{j=0}^{N+1} x_{j,i,k}, \quad \forall\ i = 1, \ldots, N,\ k = 1, \ldots, K \tag{3.6}$$

$$\sum_{j=0}^{N+1} z_{i,j,l} = \sum_{j=0}^{N+1} z_{j,i,l}, \quad \forall\ i = 1, \ldots, N,\ l = 1, \ldots, L \tag{3.7}$$

Those constraints imply that if route $k/l$ visits the location of order $i$ then it must leave it again. Those flow constraints are only applied to the orders, not the depots 0 and $N+1$. To control the flow to and from the out-depot 0 and the in-depot $N+1$, we need some additional constraints. To guaranty that every route leaves the out-depot and returns to the in-depot we have the following constraints.

$$\sum_{j=0}^{N+1} x_{0,j,k} = \sum_{i=0}^{N+1} x_{i,(N+1),k} = 1, \quad \forall\ k = 1, \ldots, K \tag{3.8}$$

$$\sum_{j=0}^{N+1} z_{0,j,l} = \sum_{i=0}^{N+1} z_{i,(N+1),l} = 1, \quad \forall\ l = 1, \ldots, L \tag{3.9}$$

We also have constraints to guaranty that routes do not return to the out-depot nor leave the in-depot.

$$\sum_{j=0}^{N+1} x_{(N+1),j,k} = \sum_{i=0}^{N+1} x_{i,0,k} = 0, \quad \forall\ k = 1, \ldots, K \tag{3.10}$$

$$\sum_{j=0}^{N+1} z_{(N+1),j,l} = \sum_{i=0}^{N+1} x_{i,0,l} = 0, \quad \forall\ l = 1, \ldots, L \tag{3.11}$$

These constraints are all very necessary for the proper arrangement of the routes.

### 3.3.2 Capacity constraints

Like in the model of the VRP we have capacity constraints. The size of each order is given in the data and is here denoted $q_i$. Each vehicle has a fixed capacity and that along with the sizes of the orders, needs to be taken into consideration when the routes are constructed.

We have made the assumption that there are two separate fleets of vehicles, one for pickup and another for delivery. The vehicles within each fleet are also assumed to be homogeneous, however, we choose to allow the possibility of vehicles of different fleets being of different capacity. In order to include that in our model we introduce two different constants noting the capacity of vehicles, $ck$ for the capacity of vehicles on pickup routes and $cl$ for the capacity of vehicles on delivery routes.

$$\sum_{i=1}^{N} q_i y_{i,k} \leq ck, \quad \forall\ k = 1, \ldots, K \tag{3.12}$$

$$\sum_{i=1}^{N} q_i v_{i,l} \leq cl, \quad \forall\ l = 1, \ldots, L \tag{3.13}$$

### 3.3.3 Number of routes

In this model we set the maximum possible number of routes to a large constant $K$ for pickup routes and $L$ for delivery routes. In fact we could set it to infinity, however, in the optimal solution there will only be use for a very limited number of routes. The number of routes is actually being optimized here with respect to the driving time, or that is the optimization of the driving time will give us the number of routes that result in the minimum driving time. That number is although not necessarily the minimum number of routes possible.

Since we do not have a constant number of routes driven, we will need a method of finding it because some factors in the model demand that. For example, we will need to know the total number of routes in the constraints for multiple use of vehicles later in this chapter.

To find the number of driven routes we somehow need to be able to count them, or the routes not driven. What we do, is we make a constraint forcing every route to leave the out-depot. The ones not driven should only go from out-depot to in-depot. Constraints 3.14 and 3.15 make sure every route leaves the out-depot.

$$\sum_{j=1}^{N+1}\sum_{k=1}^{K} x_{0,j,k} = K \tag{3.14}$$

$$\sum_{j=1}^{N+1}\sum_{l=1}^{L} z_{0,j,l} = L \tag{3.15}$$

These constraints also set $x_{0,(N+1),k}$ and $z_{0,(N+1),l}$ equal to 1 on the routes not driven, so that they go straight from out-depot to in-depot. When we then sum up the routes going from out-depot to other locations than the in-depot we get the number of routes driven, $D$ for pickup routes and $E$ for delivery routes.

$$\sum_{j=1}^{N}\sum_{k=1}^{K} x_{0,j,k} = D \tag{3.16}$$

$$\sum_{j=1}^{N}\sum_{l=1}^{L} z_{0,j,l} = E \tag{3.17}$$

Those are then used where we need the number of routes that are in fact driven, for pickup and delivery routes respectively.

### 3.3.4 Time constraints

In this problem we have limits on the working time of the vehicles. There is a time window in which the vehicles can be used. This time window is defined by the earliest possible departure time of a vehicle from the depot, along with the latest possible arrival time of a vehicle back to the depot, respectively. We need to make sure that each order is both picked up and delivered in this time window. Therefore we introduce time labels on the location of each order. The time labels indicate the time when the vehicle on the route covering it, is ready to leave the location.

In order to make these time labels we have to consider some time related factors both constant and variable.

For instance the service time $L_i$ at each customer $i$ which is the time it takes to load/unload the vehicle. It is constant and is also assumed to be identical at both pickup and delivery locations. It consists of a constant time $S$ for each stop, representing the arrival and preparation of the truck for loading or unloading. In addition we have extra time dependent on the number of pallets loaded/unloaded at that location, the time per pallet being $P$. We therefore get:

$$L_i = S + q_i \cdot P \qquad \forall\ i = 1, \dots, N. \tag{3.18}$$

It is also necessary to have the total service time defined for each route at the cross docking station, since the orders on that route can not be delivered before they have been unloaded at the cross docking station and loaded onto the vehicle driving the delivery route. This time is calculated in a similar way for both the loading and unloading and is denoted $TL_k$ for unloading of pickup route $k$ and $RL_l$ for loading on delivery route $l$.

$$TL_k = \sum_{j=1}^{N} (y_{i,k} \cdot q_i) \cdot P + S \cdot (1 - x_{0,(N+1),k}), \quad \forall\ k = 1, \dots, K \tag{3.19}$$

$$RL_l = \sum_{j=1}^{N} (v_{i,l} \cdot q_i) \cdot P + S \cdot (1 - z_{0,(N+1),l}), \quad \forall\ l = 1, \dots, L \tag{3.20}$$

Equations 3.19 and 3.20 indicate the time it takes to load or unload each order on route $k$ or $l$, depending on their sizes $q_i$ and moreover, the constant time for each vehicle to be served, namely $S$ for each route driven. Here $x_{0,(N+1),k}$ and $z_{0,(N+1),l}$, which equal 1 if the route in question is not driven, are used to control whether the constant $S$ is added or not. When the route is not driven there is no service time for it and $S$ is multiplied with 0.

In order to handle the time labels we also need to define the finishing time of pickup route $k$ at the depot, $T_k$, and the starting time at depot of delivery route $l$, $R_l$.

The maximum working time for vehicles is given in the data for the problem. We introduce the maximum working time $T_{max}$ for pickup routes and $R_{max}$ on delivery routes. Since in this case we only have a maximum working time for vehicles for the whole problem, $R_{max}$ really is the only time limit we set. The maximum working time for the pick up route $T_{max}$ is set to this same time, but this is only done to have a value to use in the constraints concerned with multi use of vehicles and does not really do much. The only reason we define these two different values is to have the possibility of limiting the working time for vehicles on pickup routes specifically.

### The total service time

The total service time is based on the service times mentioned above, in equations 3.18, 3.19 and 3.20. The total service time $TotSerTime$ can vary a bit depending on the solution. However, these variations are only caused by difference in the number of routes, since we have the constant service time $S$ at the cross docking station for each route driven.

$$TotSerTime = 2 \cdot \sum_{i=1}^{N} L_i + \sum_{k=1}^{K} TL_k + \sum_{l=1}^{L} RL_l \tag{3.21}$$

**Time constraints for the pickup routes**

The time labels used for pickup routes are denoted $t_{i,k}$ and show the time when a vehicle on pickup route $k$ has been loaded and is ready to leave the location of $i$. Then the time constraints for the pickup routes can be defined as the following.

$$x_{i,j,k}(t_{i,k} + A_{i,j} + L_j - t_{j,k}) \leq 0, \quad \forall\ i = 0, ..., N;\ j = 1, ..., N;\ k = 1, ..., K \tag{3.22}$$

$$x_{i,(N+1),k}(t_{i,k} + A_{i,(N+1)} + TL_k - T_k) \leq 0, \quad \forall\ i = 1, ..., N;\ k = 1, ..., K \tag{3.23}$$

The first constraints, 3.22, sets the time labels for the orders $j$ which are picked up after orders $i$ on routes $k$. It is set by adding the driving time between the locations of orders and the service time for order $j$ to the time label we have on $i$. The latter constraint sets the time $T_k$ which is the finishing time of pick-up route $k$ at the depot/cross-docking station.

We also have constraints to keep the time labels at 0 for routes not driven and locations of orders not visited on the route:

$$t_{i,k} \leq M \cdot y_{i,k}, \quad \forall\ i = 1, ..., N; \quad k = 1, ..., K \tag{3.24}$$

Here the time labels $t_{i,k}$ are set to 0 when order $i$ is not on route $k$, or that is if $y_{i,k}$ equals 0. This is done because the time constraints above do not specify any values for the time labels on these routes. Without these constraints those time labels could take other values than 0, and thereby make it impossible to use them like for example in the constraints concerning multi use of vehicle, found in section 3.3.5. There the time labels are used when assigning routes to vehicles. Furthermore we have similar constraints to keep the finishing time of routes not driven at 0:

$$T_k \leq M \cdot (1 - x_{0,(N+1),k}), \quad \forall\ k = 1, ..., K \tag{3.25}$$

### Time constraints for the delivery routes

We have the same kind of time labels on delivery routes, noted $r_{i,l}$. The time constraints for the delivery routes are the following.

$$z_{0,j,l}(R_l + RL_l + B_{0,j} + L_j - r_{j,l}) \leq 0, \quad \forall\ j = 1, ..., N; \quad l = 1, ..., L \tag{3.26}$$

$$z_{i,j,l}(r_{i,l} + B_{i,j} + L_j - r_{j,l}) \leq 0, \quad \forall\ i = 1, ..., N; \quad j = 1, ..., N; \quad l = 1, ..., L \tag{3.27}$$

$$z_{i,(N+1),l}(r_{i,l} + B_{i,(N+1)} - r_{(N+1),l}) \leq 0, \quad \forall\ i = 1, ..., N; \quad l = 1, ..., L \tag{3.28}$$

$$r_{(N+1),l} \leq R_{max}, \quad \forall\ l = 1, ..., L \tag{3.29}$$

The first constraints, 3.26, set the time label for the first location visited on route $l$. The second ones, 3.27, do so for the following locations and the third, 3.28, set the time labels $r_{(N+1),l}$ which is the time route $l$ is finished. Finally constraints 3.29 are put on the finishing time of the route, since it has to be within the maximum working time of the vehicle.

Like for pickup routes we also have constraints to keep the time labels at 0 for routes not driven and locations not visited on the route:

$$r_{i,l} \leq M \cdot v_{i,l}, \quad \forall\ i = 1, ..., N; \quad l = 1, ..., L \tag{3.30}$$

Since the final time label is not included in these constraints we have special constraints for it:

$$r_{(N+1),l} \leq M \cdot (1 - z_{0,(N+1),l}), \quad \forall\ l = 1, ..., L \tag{3.31}$$

In addition there are also similar constraints to keep the starting time of delivery routes at 0 for routes not driven:

$$R_l \leq M \cdot (1 - z_{0,(N+1),l}), \quad \forall\ l = 1, ..., L \tag{3.32}$$

#### Subtour elimination and linearity

In the VRP model in section 2.2 we need to use subtour elimination constraints or some constraints imposing connectivity of the routes created. Here we do not need to make any specific subtour elimination constraints because the time constraints we have defined for both pickup and delivery routes, impose connectivity of the routes. As a matter of fact, the time constraints in them self eliminate subtoures by setting time labels on each order location on each route from out-depot to in-depot.

As can be seen, the time constraints for both pickup and delivery routes are nonlinear. Therefore they had to be made linear when solving the model with exact methods. How that is done will be explained in section 4.1

#### Pickup guaranty

Having the time constraints above, an important restriction is still missing. The delivery route delivering a certain order can not start until the order has been picked up and has arrived at the depot. To guaranty this we use the following equation.

$$T_k \leq R_l + M(-y_{i,k} - v_{i,l} + 2), \quad \forall\ i = 1, ..., N; \quad k = 1, ..., K; \quad l = 1, ..., L \tag{3.33}$$

This implies that if order $i$ is on pickup route $k$ and also on delivery route $l$, or that is $y_{i,k}$ and $v_{i,l}$ are both equal to 1, then $R_l$, the starting time at depot of delivery route $l$, must be greater or equal to $T_k$ which is the finishing time of pickup route $k$ at depot. If order $i$ is neither on pickup route $k$ nor on delivery route $l$ or only on one of them, then the very large number $M$ makes this constraint meaningless.

### 3.3.5 Multiple use of vehicles

In the solution to this problem the number of routes generated should be greater than the number of vehicles used. Therefore each vehicle can probably drive more than one route.

In this section we discuss how we came up with constraints to organize the order of routes for each vehicle and to determine the number of vehicles needed. Then those constraints are described both for pickup and delivery.

#### The original constraints

Originally we got these constraints from an article [3] and adjusted them to our problem. How we did that is shown here below.

In these constraints for pickup routes, the binary variable $\alpha_{k,k'}$ is used to show whether pickup route $k$ followed by pickup route $k'$ are both assigned to the same vehicle or not. To set $\alpha_{k,k'}$ the following constraints are used.

$$\frac{t_{0,k'} - T_k}{T_{max}} + \varepsilon \leq \alpha_{k,k'} \leq 1 + \frac{t_{0,k'} - T_k}{T_{max}}, \quad \forall\ k = 1, ..., K;\ k' = 1, ..., K;\ k \neq k' \tag{3.34}$$

Here $\varepsilon$ is an infinitesimal positive number needed to enforce $\alpha_{k,k'}$ to 1 if route $k'$ starts at the same time route $k$ finishes. These constraints are shown with our variables in them, $T_k$ being the finishing time of pickup route $k$ at the depot and $t_{0,k'}$ the starting time of the vehicle on route $k'$. Consequently $T_k$ can never be greater than $t_{0,k'}$ in order for pickup route $k$ to be followed by pickup route $k'$ on the same vehicle.

We also made similar constraints for multi use of vehicles on delivery routes. Here $\beta_{l,l'}$ shows whether delivery route $l$ followed by route $l'$ are assigned to the same vehicle or not. To set $\beta_{l,l'}$ we use these constraints:

$$\frac{R_{l'} - r_{(N+1),l}}{R_{max}} + \varepsilon \leq \beta_{l,l'} \leq 1 + \frac{R_{l'} - r_{(N+1),l}}{R_{max}}, \quad \forall\ l = 1, ..., L;\ l' = 1, ..., L;\ l \neq l' \tag{3.35}$$

These constraints are different from equation 3.34 in the way that here $R_{l'}$ is the starting time at depot of delivery route $l'$ and $r_{(N+1),l}$ is the time when vehicle on route $l$ has returned back to the depot.

### An improvement of the constraints

It is obvious that only one route can be driven directly after another one. Nevertheless, when taking a closer look at the original constraints we saw that a route finishing before another one starts is by these constraints forced to be directly in front of the latter one. This does not make sense since one route can only be driven by one vehicle and only directly before one particular route and there may be many routes that are in fact finished before that route begins.

Therefore, we had to change these constraints a bit. Making the changes we kept in mind that we also had to prevent that the routes not driven would interfere with the arrangement of routes on vehicles. The constraints we ended up with do not force the routes to be assigned to the same vehicles, but they make it possible. Penalizing the number of vehicles used in the objective function will then encourage each vehicle to take more than one route. The improved constraints are introduced below for pickup and delivery respectively, along with some additional constraints.

### Multi-use-of-vehicles constraints for pick-up routes

As before we use the binary variable $\alpha_{k,k'}$ to show whether pickup route $k$ followed by pickup route $k'$ are both assigned to the same vehicle or not. To set $\alpha_{k,k'}$ we now use these constraints.

$$\alpha_{k,k'} \leq 1 + \frac{t_{0,k'} - T_k}{T_{max}} - x_{0,(N+1),k}, \quad \forall\ k = 1, ..., K;\ k' = 1, ..., K;\ k \neq k' \tag{3.36}$$

Here $T_k$ is the finishing time of pickup route $k$ at the depot and $t_{0,k'}$ is the time the vehicle on route $k'$ leaves the depot. Therefore $T_k$ can never be greater than $t_{0,k'}$ in order for pickup route $k$ to be followed by pickup route $k'$ on the same vehicle. By subtracting $x_{0,(N+1),k}$, we guaranty that routes not driven will not be scheduled in front of other routes, but that is necessary because their finishing time is always 0.

To make sure there is only one route $k'$ following route $k$ directly, the following constraints are set.

$$\sum_{k'=1}^{K} \alpha_{k,k'} \leq 1, \quad \forall\ k = 1, ..., K \tag{3.37}$$

Furthermore we have constraints that guaranty that the vehicle driving route $k'$ does not drive more than one route $k$ directly before driving route $k'$.

$$\sum_{k=1}^{K} \alpha_{k,k'} \leq 1, \quad \forall\ k' = 1, ..., K \tag{3.38}$$

Now it is possible to determine $a$, the number of vehicles needed on pickup routes, by subtracting the number of times you can reuse a vehicle from the total number of driven routes $D$ in pickup.

$$a = D - \sum_{k=1}^{K} \sum_{k'=1}^{K} \alpha_{k,k'} \tag{3.39}$$

Here we have the possibility of minimizing the number of vehicles used, by for example penalizing it in the objective function or introducing an extra objective function. By introducing an extra objective function we then have a multi objective optimization problem.

**Multi-use-of-vehicles constraints for delivery routes**

The constraints we have for multi use of vehicles for delivery routes are similar to the constraints for multi use of vehicles for pick up routes. Here $\beta_{l,l'}$ shows whether delivery route $l$ followed by $l'$ is assigned to the same vehicle or not. To set $\beta_{l,l'}$ we use these constraints:

$$\beta_{l,l'} \leq 1 + \frac{R_{l'} - r_{(N+1),l}}{R_{max}} - z_{0,(N+1),l}, \quad \forall\, l = 1, ..., L;\; l' = 1, ..., L;\; l \neq l' \tag{3.40}$$

These constraints are different from equation 3.34 in the way that here $\beta_{l,l'}$ can be set to 1 if $R_{l'} = r_{(N+1),l}$. Here $R_{l'}$ is the starting time at depot of delivery route $l'$ and $r_{(N+1),l}$ is the time when vehicle on route $l$ has returned back to the depot. That is, if the vehicle driving route $l$ has finished the route when route $l'$ starts, then it can possibly drive both routes.

Furthermore, we also have constraints exactly like the ones for pickup routes, where we make sure there is only one route $l'$ following route $l$ directly.

$$\sum_{l'=1}^{L} \beta_{l,l'} \leq 1, \quad \forall\, l = 1, ..., L \tag{3.41}$$

In addition we have constraints that guaranty that the vehicle driving route $l'$ does not drive more than one route $l$ directly before driving route $l'$, are also necessary for the delivery part.

$$\sum_{l=1}^{L} \beta_{l,l'} \leq 1, \quad \forall\, l' = 1, ..., L \tag{3.42}$$

The number of vehicles on delivery routes $b$ can also be calculated the same way it is done for pickup routes, since we can now easily add up the number of times each vehicle is used.

$$b = E - \sum_{l=1}^{L} \sum_{l'=1}^{L} \beta_{l,l'} \tag{3.43}$$

## 3.4 The possibility of not reloading at the cross docking station

In the problem description we discussed the possibility of having one vehicle driving both a pickup and a delivery route and leaving some of the orders on the vehicle instead of reloading them all at the cross docking station. This has not been implemented in this model. As mentioned before we decided to assume we had two separate fleets, one for pickup and another one for delivery. Having two fleets made it a little more complicated to introduce the possibility of using the same a vehicle for both pickup and delivery. Because the model was already very large and there is a time limit on this project since it is a Master's Thesis, we decided to omit this possibility.

# Chapter 4

# Solution of the mathematical model

To solve the mathematical model with exact methods we formulated it in GAMS. We decided to solve the model with exact methods in order to be able to use the LP relaxation of the model to get a lower bound. The solution we get from applying the metaheuristic can then be compared with this lower bound to give us an idea of how good that solution is.

As well as solving the model to get a lower bound there is also another good reason for doing so, namely to make sure the model does exactly what it is supposed to do. Solving the problem with exact methods did in fact prove to be a good idea to make sure the model worked correctly, because when we started formulating it in GAMS we came up with quite a few faults and shortages of our original model that we had not foreseen. This really was not all that surprising though since the model of this problem is very big and complex and that in it self makes it easy to overlook some details. Moreover, solving the mathematical model in GAMS lead to some improvements of the model as well as adding a few constraints to the model. It was also necessary to add a few constraints in order to use in the GAMS model, that will be discussed in more detail in the following section.

## 4.1 Adjusting the constraints

To be able to formulate the mathematical model in GAMS we need to make adjustments to some of the constraints.

To begin with, the time constraints need to be linearized because they are nonlinear in the mathematical model and therefore not applicable in the linear solution. Those are constraints 3.22 - 3.28.

To linearize the constraints we use big $M$ which is a suitable large number. Instead of multiplying with $x_{i,j,k}$ or $z_{i,j,l}$ to control whether the constraints are active or not, we subtract those variables from 1 and multiply the outcome with $M$. By doing so we are no longer multiplying variables with variables, nevertheless the constraints are working in the exact same manner as before. The linearized time constraints for pickup are shown in equations 4.1 and 4.2.

$$t_{i,k} + A_{i,j} + L_j - t_{j,k} \leq M \cdot (1 - x_{i,j,k}), \quad \forall\, i = 0, ..., N;\ j = 1, ..., N;\ k = 1, ..., K \quad (4.1)$$

$$t_{i,k} + A_{i,(N+1)} + TL_k - T_k \leq M \cdot (1 - x_{i,(N+1),k}), \quad \forall\, i = 1, ..., N;\ k = 1, ..., K \quad (4.2)$$

The linearized time constraints for delivery are shown in equations 4.3 - 4.5.

$$R_l + RL_l + B_{0,j} + L_j - r_{j,l} \leq M \cdot (1 - z_{0,j,l}), \quad \forall\, j = 1, ..., N;\ l = 1, ..., L \quad (4.3)$$

$$r_{i,l} + B_{i,j} + L_j - r_{j,l} \leq M \cdot (1 - z_{i,j,l}), \quad \forall\, i = 1, ..., N; j = 1, ..., N;\ l = 1, ..., L \quad (4.4)$$

$$r_{i,l} + B_{i,(N+1)} - r_{(N+1),l} \leq M \cdot (1 - z_{i,(N+1),l}), \quad \forall\, i = 1, ..., N;\ l = 1, ..., L \quad (4.5)$$

In addition we also had to add some constraints on $y_{i,k}$ and $v_{i,l}$ to the model when formulating it in GAMS. These binary variables are set to 1 if order $i$ is picked up on route $k/l$ and in the mathematical model they are only defined for the locations of orders. Because we use the parameter $i$ just like we do in the mathematical model and define $y_{i,k}$ and $v_{i,l}$ as variables depending on $i$, we need to make sure $y_{0,k}$ and $y_{(N+1),k}$, along with $v_{0,l}$ and $v_{(N+1),l}$ respectively are set to 0. Since 0 and $N+1$ represent the depot, these instances of the variables do not really exist, but are automatically created in GAMS. Therefore we added the constraints in 4.6 and 4.7.

$$\sum_{k=1}^{K} y_{0,k} = \sum_{k=1}^{K} y_{(N+1),k} = 0 \quad (4.6)$$

$$\sum_{l=1}^{L} v_{0,l} = \sum_{l=1}^{L} v_{(N+1),l} = 0 \quad (4.7)$$

## 4.2 The solution process

The final version of the mathematical model was presented in chapter 3. In this section we will discuss briefly which additions we made to the model in the process of making the proper GAMS code to use to solve the problem.

At first we had some problems getting the desired solution when using the GAMS code we had made, based on the original version of the mathematical model. To begin with, the solutions we got were inadequate because they did not give any appropriate routes. Therefore the model obviously lacked something. When taking a closer look we discovered that our model did in fact need some additional constraints.

### 4.2.1 The routes

The first problem we discovered was that we did not have any way to enable us to keep track of which routes, and thereby how many routes, were actually used in the solution. However, that is very necessary for the model to function correctly.

In the model the number of routes possible is not the actual number of routes driven, as the number of routes is to be determined by the solution. We only assume an upper bound $K$ on the number of routes driven. Therefore we need something to indicate which of these $K$ routes are in fact driven in a particular solution. We solved this by setting $x_{0,(N+1),k}$ and $z_{0,(N+1),l}$ equal to 1 for routes not driven, thereby also getting the total number of routes not driven by summing those variables up. How this is done is described back in section 3.3.3.

### 4.2.2 Inserting the depot

Having solved the problem with the routes, we discovered another problem. We were still not getting a valid solution because the routes did not go from the out-depot to the in-depot like planed. We realized that we needed to have some special constraints forcing all routes to go out of the out-depot and in to the in-depot, as well as not into the out-depot nor out of the in-depot. These are constraints 3.8 - 3.11, found in section 3.3.1.

### 4.2.3 Improvements of the time constraints

After making the additions to the model discussed above, we managed to get a solution that gave us the correct route configuration. But still the solution was not quite right. Another problem we ran into was that the time labels were not functioning correctly. We got the right time labels for orders picked up or delivered on each route, however there was nothing forcing the time labels to be equal to zero on the routes not driven or on orders not picked up on the particular route. Those values were randomly coming up where they were not expected. This really did not do any harm in the basic solution, nevertheless it made it impossible to use the time labels in other constraints. However, the time labels are in fact used in the constraints for multiple use of vehicles found in section 3.3.5, when organizing which routes each vehicle should drive.

To solve this problem we added constraints 3.24 and 3.30 that keep the time labels at 0 for routes not driven and orders not picked up or delivered respectively. This was discussed in more detail in section 3.3.4.

### 4.2.4 Improving the multi use of vehicles constraints

When we had got the time labels right, there were still some problems with the constraints concerning multiple use of vehicles. They did not work as planed. Having taken a closer look at the original constraints we discovered that they did in fact need some improvements. How that was done has been discussed in detail back in section 3.3.5.

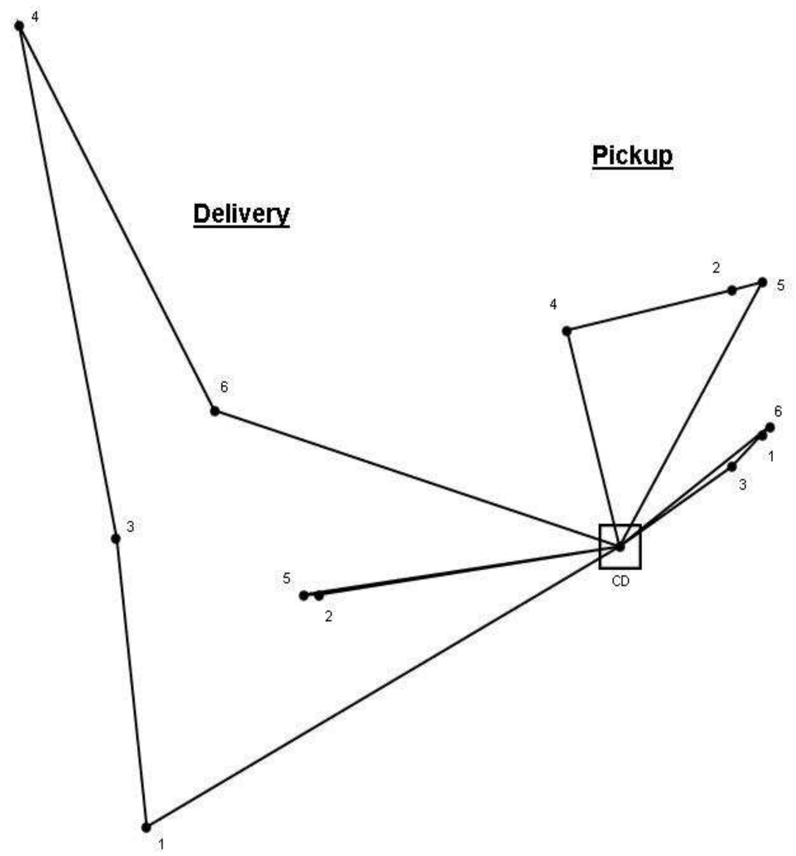


Figure 4.1: A very small problem solved with our GAMS program.

## 4.3 Testing the program

While putting together the GAMS code we used a small data set to try it out on. The data set had only 6 orders in it, 3 possible pickup routes and 3 possible delivery routes to put the orders on. However for that very small size of a problem it took about 15 hours to get an optimal solution when all the appropriate constraints had been added.

To make sure the solution we got from the GAMS program was what we had intended it to be, we plotted the solution which can be seen in figure 4.1. As can be seen in the figure, the solution consists of two pickup routes which are the ones on the right side of the figure and two delivery routes on the left side of the figure. The orders are numbered from 1 to 6 and the box is the cross docking station. When looking at the solution we can see that the routes are formed correctly and that this is most likely the optimal solution of this particular problem.

For further information, each route was driven by different vehicle in this solution, even if we penalized the number of vehicles used.

The final version of our GAMS code is shown in appendix B.

## 4.4 Problem size

With such a large mathematical model we have incredibly many constraints, variables and constants, not to mention if we would try to solve it to optimality for a data set with 247 orders like the first day in the given data. To show how large this problem really is we have put together a table showing the approximate number of constraints, variables and constants for this model when we have 6, 100, 160 and 247 orders to transport, see table 4.1. The number of pickup and delivery routes possible is here chosen appropriately with respect to the capacity of the vehicles and order sizes.

Table 4.1: Approximate number of constraints, variables and constants in the model for the various data sets

| Orders (N) | Pickup routes (K) | Delivery routes (L) | Number of constraints | Number of variables | Number of Constants |
|---|---|---|---|---|---|
| 6 | 3 | 3 | 527 | 503 | 149 |
| 100 | 40 | 40 | 1.003.928 | 851.845 | 21.017 |
| 160 | 60 | 60 | 3.752.308 | 3.195.365 | 52.817 |
| 247 | 100 | 100 | 14.940.602 | 12.519.805 | 124.505 |

Looking at these large numbers there is no doubt that we are looking at a very large scale problem, and it is clear that it is not feasible to solve this problem to optimality. Like mentioned before it took about 15 hours to solve the problem with only 6 orders, 3 possible pickup routes and 3 possible delivery routes. Keeping that in mind it is very obvious that solving the problem with 100, not to mention 247 orders, is not an feasible option.

Therefore we have chosen to solve this problem by using metaheuristics. How that was done will be discussed in the following chapter.

## 4.5 Conclusion

It is safe to say that solving the model in GAMS brought forward quite a few necessary additions to the original mathematical model. After having made all the adjustments and additions described above we ended up with a GAMS code that can be used to solve a very small problem of this kind to optimality. Furthermore, we intend to use this GAMS code to solve our problem with LP relaxation. By doing that we will then get a lower bound to compare the results of our metaheuristics with. This will be discussed in chapter 8.

# Chapter 5

# Solution Approach

To solve a problem of this size we need to use metaheuristics. As has been shown in chapter 4, using exact methods is very complicated for larger problems and is therefore not a feasible option.

In this chapter we will explain which method we choose to use for this problem and how we apply it.

## 5.1 Choosing a metaheuristic

There are several metaheuristics that could be used to solve our problem. Two were mainly considered, those are Simulated Annealing (SA) and Tabu Search (TS). They will be referred to with their abbreviation from this point.

SA is a generic probabilistic meta-algorithm for the global optimization problem. It depends on random steps to escape local optima and locates a good approximation to the global optimum of a given function in a large search space. It has been used on the VRP with good results [7].

TS belongs to the class of local search techniques. It uses memory structures to enhance the performance of the local search method by avoiding cycling and to orient the search toward unexplored region of the solution space. TS has proved to give very good results when applied on the VRP problem [7].

For large neighborhoods TS may not be the ideal method since the neighborhoods could be too large to be searched exhaustively. The problem to be solved has a quite large neighborhood so we decided to use SA. SA is the one of the simplest metaheuristic algorithms and is quite simple to implement. It has only few parameters which we will come to later.

## 5.2 Simulated Annealing

The simulated annealing algorithm is inspired from annealing in metallurgy. Annealing is a technique which involves heating and controlled cooling of materials. By cooling a heated metal it freezes into a regular crystalline structure and thereby increases its strength. The heat causes the atoms to become unstuck from their initial positions and wander through the states of higher energy. The slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

In an analogous way, a simulated annealing algorithm searches for the optimal solution. It moves randomly in the solution space looking for a solution that minimizes the value of some objective function. The moves the algorithm makes are random and therefore the value for a given objective function may decrease, increase or even stay the same.

Table 5.1 shows how standard simulated annealing terminology relates to our problem [4].

Table 5.1: Terminology relationship

| SA terminology | Problem equivalent | Purpose/definition |
|---|---|---|
| System | Routing problem | To be reconfigured through the annealing |
| Energy state | Configuration | An arrangement of pickup and delivery routes |
| Energy | Configuration cost | Total cost of the system as configured |
| Atoms/particles | Pickup routes, delivery routes | Routes are randomly changed during the search for their lowest cost arrangement |
| Temperature | Temperature | Influences the acceptance probability for inferior solutions |
| Cooling | Cooling | Decreases probability of accepting |
| Perturbation | Perturbation | A change in configuration |

The SA algorithm always accepts moves that lower the value of the objective function. However, unlike local search descent methods which do not allow non-improvement moves at any iteration, SA accepts non-improving solutions with the probability:

$$p = \exp(-\delta/T). \tag{5.1}$$

This is the Boltzmann distribution where $T$ is the temperature and $\delta$ is the change in the value of the objective function. The temperature starts high and decreases over time toward zero. When $T$ is high the algorithm is in an exploratory phase also called a random walk and the probability of accepting a bad move is high, whereas when $T$ is low it has reached an exploitation phase or a randomized hill climbing and when $T$ is close to 0 the probability of accepting a new move is close to 0.

The pseudocode for SA is presented in Algorithm 1. First an initial solution is generated in line 1. In line 4 a new neighborhood solution is found randomly and in line 5 the change in the value of the objective function is calculated. The new solution is now accepted either, if $\delta$ is negative which tells us that the new objective value found is better than the old one, or with probability $p$, given in equation 5.1, if $\delta$ is positive. If a solution is not accepted, a new neighborhood solution is randomly generated and so on.

There are two variations of the cooling function for SA. One is called the inhomogeneous version of SA where the temperature is lowered in every iteration and the second version is called homogeneous where the temperature is kept constant for a number of iterations.

The temperature is lowered according to a cooling schedule. It begins at an initial temperature $T_{start}$ and decreases in steps according to

$$T_{k+1} = T_k \cdot \alpha, \quad \text{where} \quad 0 < \alpha < 1. \tag{5.2}$$

**Algorithm 1** Simulated Annealing

select initial solution $s_0$
$t = T_{start}$
**repeat**
  select $s \in N(s_i)$
  $\delta = f(s) - f(s_i)$
  **if** $\delta < 0$ OR with probability $p(\delta, t_i)$ **then**
    $s_{i+1} = s$
  **end if**
  $t_{i+1} = t_i \cdot \alpha$
**until** stopping criterion is true

The algorithm terminates when the stopping criterion is met which is typically when the temperature has reached some final value $T_{end}$.

## 5.3 Neighborhood search methods

Choosing neighborhood search methods is just as important as choosing a metaheuristic, since there are numerous ways to search the neighborhoods and it is important to choose methods which are suitable and right for the structure of the problem. We propose two neighborhood search algorithms for this problem. The first one we have chosen to call **N-A1** and it is based on a swapping method. The second one is based on a re-insert method and referred to as **N-A2** in the following sections.

### 5.3.1 Neighborhood algorithm N-A1

The neighborhood algorithm **N-A1** is based on a swapping method where two orders in different routes are swapped [1], see Fig. 5.1.

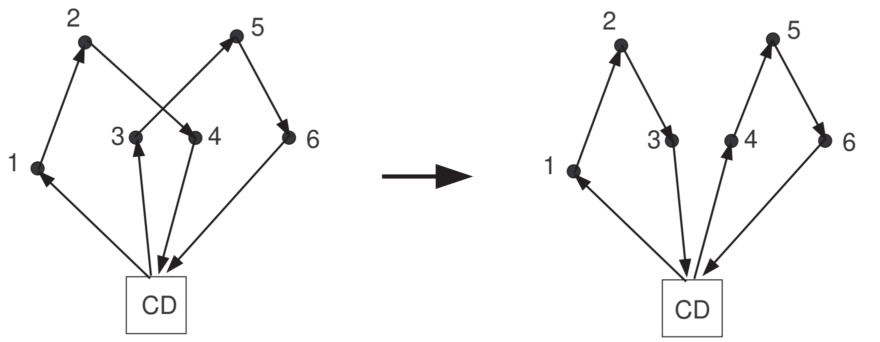


Figure 5.1: A small example of swapping between routes. The figure on the left shows the routes before the swap and the figure on the right after the swap.

In the following sections we will represent the routes as vectors, see Fig. 5.2. A route starts at the depot which is denoted with 0. The second index of the vector denotes the order which should be picked up or delivered after leaving the depot. The third index indicates the following order and so on until the route ends at the depot again.

The example in Fig. 5.1 is illustrated in Fig. 5.2 with vectors as have been described above.

$$\begin{array}{lcl} 1 : [\ 0\ 1\ 2\ 4\ 0\ ] & & 1 : [\ 0\ 1\ 2\ 3\ 0\ ] \\ & \longrightarrow & \\ 2 : [\ 0\ 3\ 5\ 6\ 0\ ] & & 2 : [\ 0\ 4\ 5\ 6\ 0\ ] \end{array}$$

Figure 5.2: Vector representation of swapping orders 3 and 4. Left side shows the routes before the swapping and the right side after the swapping.

The pseudocode for **N-A1** is presented in Algorithm 2.

**Algorithm 2** Neighborhood algorithm N-A1

$j = 0$
**while** swapFeasible == false AND $j < J$ **do**
    pick an order $cID_1$ randomly
    pick another order $cID_2$ randomly from a different route
    **if** swapFeasible == true **then**
        swap $cID_1$ and $cID_2$
    **end if**
    $j = j + 1$
**end while**

The algorithm picks two orders at random from two different routes and checks if swapping them is a feasible move. The algorithm terminates when a feasible solution is found or when it has tried to find a new feasible solution $J$ number of times. If the algorithm has not found a feasible solution when it terminates it returns the original solution.

Using the swapping method alone does not allow us to decrease the total number of routes or the number of orders within each route. However, in algorithm **N-A2** it is possible for the number of routes to decreases. Furthermore, the number of orders on a route could change.

### 5.3.2 Neighborhood algorithm N-A2

We propose another neighborhood search based on a re-insert method which picks an order on a given route and inserts it into another [1], see Fig. 5.3.

The vector representation of the re-insert method can be seen in Fig. 5.4. As can be seen in the figure, the number of orders in routes can increase and decrease. Furthermore, if we would apply the re-insert method on a route 1 with only one order and another route 2

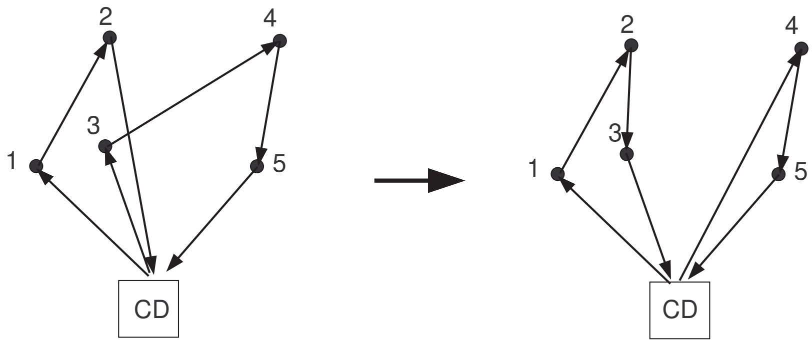


Figure 5.3: A small example of re-inserting an order into a route. The figure on the left displays the routes before the move and on the right shows the routes after the move.

| | | |
|---|---|---|
| 1 : [ 0 1 2 0 ] | $\longrightarrow$ | 1 : [ 0 1 2 3 0 ] |
| 2 : [ 0 3 4 5 0 ] | | 2 : [ 0 4 5 0 ] |

Figure 5.4: Vector representation of the re-insert method. Order 3 in route 2 is inserted into route 1. On the left we see the routes before the insert and on the right after the insert.

with one or more orders and inserting the order on route 1 into route 2 would be feasible then route 1 would be deleted.

The pseudocode for the Re-Insert method can be seen in Algorithm 3. The re-insert

**Algorithm 3** Neighborhood algorithm N-A2

$j = 0$
**while** reInsertFeasible == false AND $j < J$ **do**
  pick an order $cID_1$ randomly
  get route ID $rID_1$ for $cID_1$
  pick a route randomly $rID_2$ which is different from $rID_1$
  **if** reInsertFeasible == true **then**
    remove order $cID_1$ from route $rID_1$
    add order $cID_1$ to route $rID_2$
  **end if**
  $j = j + 1$
**end while**

algorithm randomly picks an order and then picks a route different from the route the picked order is in. It checks if it is feasible to insert the order in the new route. If the merge is feasible the algorithm terminates and it returns the new feasible solution. Furthermore, **N-A2** terminates after $J$ attempts to find a new feasible solution and returns the original solution if a new one has not been found already.

Not only does the re-insert algorithm make it possible to decrease the total number of routes, but also it makes it possible to increase and decrease the number of orders in the routes.

## 5.4 Implementation of SA

All algorithms were implemented using C++ and computational experiments carried out on a PC with a 958 MHz AMD Athlon(tm) processor with 256 MB RAM, running a Windows XP operating system. The source code can be found at:
`http://www.student.dtu.dk/ s010347/SourceCode.zip`.

With the two neighborhood algorithms we have made two versions of the SA algorithm. The difference lies in the neighborhood solution search. The first version, **SA-A1**, uses **N-A1** to search for new neighborhood solutions whereas the second one, **SA-A2** uses **N-A1** as well as **N-A2**. The second algorithm, **SA-A2**, is in fact an improvement of **SA-A1** since the number of routes or the number of orders in a route can not be changed in **SA-A1**.

The first version will be referred to as **SA-A1** and the second one as **SA-A2** in the following chapters and sections.

In the process of implementing the algorithms we started by implementing the pickup part only, adding the delivery part later on. The pickup part is slightly more simple than the delivery part since we chose to have no time limit on when the orders have to be at the cross dock after being picked up. In the delivery part we need to take into account that the delivery trucks cannot start their routes until all the orders on their route have been picked up and have arrived at the cross dock. Furthermore, the trucks have a time limit for when they need to be back at the depot after delivering the orders on their routes which also makes the delivery part a bit more complicated.

### 5.4.1 SA algorithm with N-A1: SA-A1

Our first version of the simulated annealing algorithm uses the neighborhood solution algorithm which uses swapping to find a new solution. The pseudocode for this algorithm, **SA-A1**, can be seen in Algorithm 4. The algorithm starts by selecting a feasible initial solution and initializing the temperature. In line 5 a loop starts which does not stop until a certain stopping criterion is true. This could be a number of iterations, time or even when the temperature has reached a specific value. Our algorithm is homogeneous and therefore a while loop has been added. The pseudocode for the general SA, see Algorithm 1, is an inhomogeneous version of the algorithm. As stated in a previous section, the homogeneous version iterates for a number of times before lowering the temperature. In **SA-A1** it iterates for $J$ number of times before lowering the temperature. In each iteration it starts by selecting a new neighborhood solution with **N-A1**. After calculating the change in the value of the objective function it either accepts or rejects the new solution. We generate a random number $r$ on UD(0,1), the uniform distribution from 0 to 1. If if $r \leq p$ the solution is accepted else not. $p$ is the Bolzmann probability, see Eq. 5.1.

**Algorithm 4** Simulated Annealing Algorithm SA-A1

read data
select a feasible initial solution $s_0$
$i = 0$
$t = Tstart$
**repeat**
  $j = 0$
  **while** $j < J$ **do**
    select $s \in N_{Swap}(s_i)$
    $\delta = f(s) - f(s_i)$
    **if** $\delta \leq 0$ **then**
      $s_{i+1} = s$
    **else**
      generate $r \in UD(0,1)$
      **if** $r \leq e^{-\delta/t_i}$ **then**
        $s_{i+1} = s$
      **end if**
    **end if**
    $j = j + 1$
  **end while**
  $t_{i+1} = t_i \cdot \alpha$
  $i = i + 1$
**until** stopping criterion is true

The solution may be viewed as a solution of two VRP problems connected with the depot, therefore we decided to apply the neighborhood search algorithm in turns on the pickup part and the delivery part respectively.

### 5.4.2 SA algorithm with N-A1 and N-A2: SA-A2

Our second version of the SA algorithm is an improvement of the first version. When searching for a new neighborhood solution we start by applying the **N-A1** algorithm for $J/2$ number of iterations and then the **N-A2** for another $J/2$ number of iterations. The pseudocode for **SA-A2** can be seen in Algorithm 5.

**Algorithm 5** Simulated Annealing Algorithm SA-A2

```
1: read data
2: select a feasible initial solution s_0
3: i = 0
4: t = Tstart
5: repeat
6:    j = 0
7:    while j < J do
8:       if j < J/2 then
9:          select s ∈ N_Swap(s_i)
10:      else
11:         select s ∈ N_Re-Insert(s_i)
12:      end if
13:      δ = f(s) − f(s_i)
14:      if δ ≤ 0 then
15:         s_{i+1} = s
16:      else
17:         generate r ∈ UD(0,1)
18:         if r ≤ e^{−δ/t_i} then
19:            s_{i+1} = s
20:         end if
21:      end if
22:      j = j + 1
23:   end while
24:   t_{i+1} = t_i · α
25:   i = i + 1
26: until stopping criterion is true
```

**SA-A2** is just like **SA-A1** except for lines 8-12 where it either uses **N-A1** or **N-A2** to search for a new solution.

As in **SA-A1** the neighborhood search is applied in turns on the pickup part and the delivery part respectively.

## 5.5 Implementation details

In this section we will explain how we sort within individual routes, what makes a solution feasible, how we generated the initial solution and the calculation of the objective value.

### 5.5.1 Sorting within the Routes

To be able to construct the routes and thereby to say whether a solution is feasible or not we need to have decided in which order to pickup or deliver the orders. To do so we use a simple method namely the nearest neighbor method.

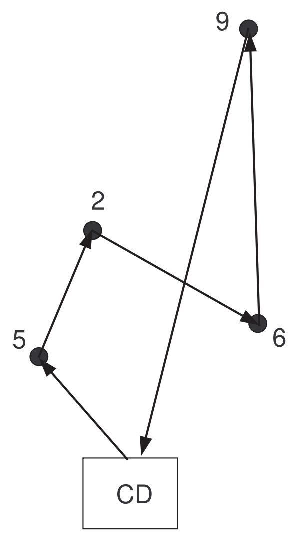


Figure 5.5: An example route constructed with nearest neighbor method

Given that we have a route which should pick up orders 2, 5, 6 and 9, we start ordering them by their distance from the depot, shortest being first and so on.

[ 0 5 6 2 9 0 ]

This tells us that we should pick up order 5 after leaving the depot. Now, to find out which order we should pick up next we sort the rest by their distance from order 5.

[ 0 5 2 6 9 0 ]

We repeat this procedure until all orders have been picked up and the truck heads back to the depot. As before, the depot is denoted with 0. For this given example we end up with the route in Figure 5.5.

### 5.5.2 Feasibility

To make sure that a solution is feasible there are two things which are to be considered. Firstly, the load on the trucks can not exceed their capacity and secondly, the delivery trucks have to be back at the cross dock before the time limit.

This means that when we change the arrangement within the pickups or deliveries we need to check if the capacity constrains are violated and if the trucks delivering the orders can still make it to the cross dock station in time.

### 5.5.3 Initial Solution

There are several methods to generate an initial solution for this problem. We implemented and used two which we will now describe.

The first one is very simple. It is simply generated by randomly selecting unassigned orders in routes. In the pickup part the orders are accepted in the routes if the capacity of the total load does not exceed the capacity of the truck. When an order can not be accepted on the route being generated we start a new route which this order is on.

When generating the delivery part we also need to take into account that there is a time limit on the trucks. So, before accepting an order in a route we need to keep in mind that a truck can not be loaded for delivery until all the orders have been picked up and unloaded from the truck at the depot. Keeping this in mind, we find the time when the truck can start delivering the orders and check whether it can make it back to the cross dock before the time limit. Subsequently, if an order is randomly picked and can not be accepted in the route we are constructing, because the total load will exceed the capacity limit or the truck could not be back in the depot before time limit, we start a new route containing this order.

The second method to generate an initial solution is a building heuristic. We start by selecting a random order and then add its nearest unassigned neighbors until the truck is fully loaded and we start a new one.

Since this is a Master's Thesis and limited time can be spent we decided to use only one of these methods when we parameter tune and test the SA algorithms. Early on, after experimenting with the methods, we decided that the second method would be used. It gives a well structured initial solution which is much better than the first one.

### 5.5.4 Objective Function

The objective is to minimize the travelling time for the trucks. As the problem is split in two, pickup and delivery, and connected with the cross dock station, the objective function simply sums up the total time for the pickup routes and the delivery routes. To illustrate

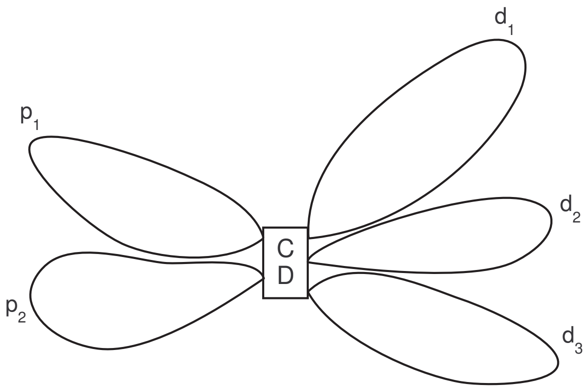


Figure 5.6: This solution has has two pickup routes and three delivery routes.

this we consider a small example, see Fig. 5.6. There we have two pickup routes, $p_1$ and $p_2$ and three delivery routes, $d_1$, $d_2$ and $d_3$. The objective function for solution $s$ is:

$$f(s) = \sum_{n=1}^{N} p_n + \sum_{m=1}^{M} d_m \tag{5.3}$$

where $N$ denotes the total number of pickup routes and $M$ the total number of delivery routes. The variable $p_n$ is the time it takes to finish route $n$, from when truck $n$ leaves the depot until it has picked up all the orders on its route and has been unloaded at the depot. The $d_m$ variable is similar. It is the time from when the loading of the truck starts until it has delivered all orders on its route and returned to the depot.

## 5.6 Summary

In this chapter we described why we chose to use simulated annealing and gave a description of the method. We proposed two neighborhood search methods and introduced two versions of the SA algorithm with different neighborhood searches. At last, we explained the implementation of the algorithms in detail.

The algorithms we have implemented give us solutions that fulfill the same requirements our mathematical model does, except for the possibility of using the same vehicle more than once.

# Chapter 6

# Improvements

In the previous chapter we presented basic versions of two very similar SA algorithms. There are, however, several ways to improve them. Since there is a time limit on our project we only considered using the two following methods to improve our algorithms.

## 6.1 Eliminating a single order route

Without any improvements on the **SA-A1** algorithm, which only uses swapping to find a neighborhood solution, the number of routes will depend solely on the initial solution. This is a great disadvantage for the algorithm. It could improve the solution if we would check if there were routes containing just a single order in the solutions. We would then try to insert this order into another route thereby decreasing the number of routes for the solution.

Although the number of routes can be decreased in algorithm **SA-A2** because it uses the re-insert method as well as swapping, inserting orders from a single order route into another and thereby decreasing the number of routes could also improve the solution.

The pseudocode for this procedure can be seen in Algorithm 6.

We start with the pickup or delivery routes, depending on which part we are applying the algorithm to. Initially we scan for routes containing a single order. If we find a route with only a single order we randomly pick another route and check if it is feasible to insert the single order into the selected route. When we apply the algorithm to the pickup or delivery part we need to make sure that the move does not violate the time constraint for the deliveries. Furthermore, the capacity constraint has to be respected.

We have chosen to call this algorithm *oneInRoute* and we apply it to the solution $s_i$ in algorithms **SA-A1** and **SA-A2** every time we lower the temperature. In their pseudocodes presented in section 5.4 we would insert the line

$s \in oneInRoute(s_i)$

after line 20 in Alg. 4 in section 5.4.1 and after line 24 in Alg. 5 in section 5.4.2.

**Algorithm 6** An algorithm to eliminate single order routes, *oneInRoute*

```
get n = total number of routes
for i = 1 to n do
    j = 0
    if only one order in route i then
        while j < J do
            randomly pick another route different from route i
            if insertFeasible == true then
                insert order from route i into the randomly picked route
                eliminate route i
            end if
            j = j + 1
        end while
    end if
end for
```

## 6.2 Improving the routes

To construct the routes we used a nearest neighbor algorithm, see Sec. 5.5.1. The nearest neighbor method does usually not give the best route. However, the solution can be improved so it gives a good solution. There are several algorithms, based on simple modifications within the routes, which can be used to improve the routes. The algorithm we use to improve the routes is the 2-Opt algorithm, the pseudocode can be seen in Alg. 7. The 2-Opt algorithm was first proposed by Croes in 1958. The version of the algorithm we use was proposed by Lin and Kernighan in 1971 and is a very good heuristic for the Euclidean TSP [6].

**Algorithm 7** Local Search 2-Opt Algorithm (Lin-Kernighan)

```
1: start with a feasible tour T
2: get N = number of orders in tour and L = length of tour
3:
4: let T* = T and L* = L
5: for i = 1 to N do
6:     for j = i+1 to N do
7:         consider swapping orders i and j
8:         if L' < L* where T' is the resulting tour after swapping then
9:             L* = L' and T* = T'
10:        end if
11:    end for
12: end for
13: if T = T*, no improvement possible any more then
14:     return T* as the best tour
15: else
16:     set T = T* and L = L* and repeat from line 4
17: end if
```

The 2-Opt algorithm is a local improvement algorithm, which can be used to convert one tour or route into another. When we apply the algorithm on a route it terminates when no further improvements can be obtained by the operation. An example of a move is displayed in Fig. 6.1 where orders 2 and 4 have been swapped. [5]

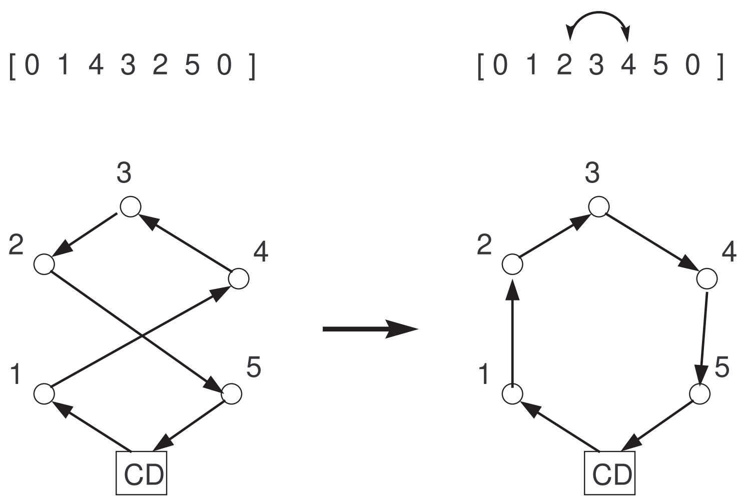


Figure 6.1: A 2-Opt move. The route on the left is the original one and the resulting one on the right. The cross docking station (CD) is denoted with 0 in the vector representation of the routes above the figures.

The algorithm, see Alg. 7, starts with a route or a tour which should be improved. It considers swapping orders $i$ and $j$ by checking if the swap leads to a shorter route. If the resulting route shortens the length, the solution is stored as a candidate for future consideration. If the swap does not improve the route the solution is discarded and the algorithm considers the next pair of orders and so on until all pairwise exchanges are considered. Each order can be exchanged with $(n-1)$ other orders. Furthermore, since there are $n$ orders in total, there are $n(n-1)/2$ different exchanges. In lines 4 to 12, see Alg. 7 these exchanges are considered. The resulting solution after line 12, is the one that shortened the length of the tour the most. If there has been an improvement in the length of the tour then algorithm repeats this procedure starting with the improved solution as the new solution. It starts to repeat from line 4 to find another better solution. If, however, a better solution can not be found the algorithm terminates returning the remaining solution as the best one. [5]

We apply this algorithm to the routes every time a route changes, that is after an order has been added or removed.

## 6.3 Summary

We improved our simulated annealing algorithms, **SA-A1** and **SA-A2**, by applying two algorithms to it. Firstly an algorithm to eliminate single order routes and secondly, an algorithm to improve the routes. Both improvement algorithms proved to lower the objective values for the solutions.

Due to the time limit on the project we did not implement other improvement algorithms. Other improvements we would consider, would allow the option of increasing the number of routes. It is possible with our algorithms that an initial solution could be generated with such a few routes that it will be almost impossible for the algorithms to improve the solution. We will leave this improvement to further work.

# Chapter 7

# Parameter Tuning

An advantage of using Simulated Annealing is that the algorithm has only few parameters. To get an appropriate value for each parameter they should be tuned. Not all parameters can be tuned though since it is a very time consuming process. Since this is a Master's Thesis and we therefore have limited time to spend on the parameter tuning, we divide the parameters into two groups, important parameters and less important parameters, and fix the not so important parameters before tuning the important ones.

In this section we describe the parameters involved and how they are tuned or otherwise determined.

## 7.1 The Parameters

The parameters can be divided into two groups. Important parameters and less important parameters. The parameters in our SA algorithms are listed in the table 7.1.

Table 7.1: The important and less important parameters in the algorithms.

| | |
|---|---|
| 1. | The initial temperature, $T_{start}$ |
| 2. | The cooling rate, $\alpha$ |
| 3. | Number of attempts to find a new feasible neighborhood solution in pickup part |
| 4. | Number of attempts to find a new feasible neighborhood solution in delivery part |
| 5. | Number of attempts to merge a pickup route which has only a single order with another pickup route |
| 6. | Number of attempts to merge a delivery route which has only a single order with another delivery route |
| 7. | Number of iterations before lowering the temperature |

Ideally all parameters should be tuned, however, it just takes too much time to make exhaustive tests on all combinations. Parameter tuning is very time consuming and in some cases perhaps not worth while. We have limited time to spend on the parameter tuning since this is a Master's Thesis project, so we choose to tune only the most important parameters involved.

The less important parameters in our algorithm, which we choose not to tune are the following:

**Parameters 3 and 4** The number of attempts to find a new feasible neighborhood solution for pickup and delivery routes respectively. This parameter is within the function which randomly generates a new feasible neighborhood solution. Through the process of implementing the function the parameter was set so that it would choose randomly new orders to swap suitably many times until a feasible solution is found.

**Parameters 5 and 6** The number of attempts to insert an order of a single order route into another route is a less important parameter and is not tuned. This parameter controls how often we try to insert a single order from a single order route into another route. A candidate route for the single order route to be merged with is randomly selected. It may not be feasible to insert the order into the selected route so a new route is selected and considered for insertion. The parameter was set high enough that it would try to insert the order into most routes.

**Parameter 7** The number of iterations before lowering the temperature is kept constant and therefore not tuned. When choosing a suitable value for the parameter we needed to keep in mind that selecting a too large value could lead to spending a lot of time searching for solutions when the temperature is high. Furthermore, selecting a too low value could lead to the algorithm not searching the neighborhood thoroughly enough for solutions. After experimenting it was set to a value we considered suitable.

One parameter not mentioned above is the termination parameter. In this case, we do not consider it to be a parameter since it controls the time the algorithm should run. Moreover, time is not a parameter since this is a case from the real world and we have to make assumptions about how long the customer would be willing to wait for a solution.

To determine the running time we considered how long it would be acceptable to wait for a solution when using the program. After some consideration it was decided that approximately 15 minutes would be an appropriate time. As a result of that, the time elapsed is checked after lowering the temperature and if the elapsed time is less than 15 minutes it iterates again else it terminates. The running time may vary from 15 minutes to 17 minutes.

## 7.2 Selecting a Cooling Schedule

Parameters **1** and **2** in table 7.1 are considered the important parameters since they are a part of the cooling schedule.

A cooling schedule is a method for controlling the temperature, $T$. The schedule involves an initial temperature and a rule for decrementing the temperature as the search progresses. The initial temperature, $T_{start}$ should be high enough, so that most moves in the beginning of the execution of the algorithm are accepted.

Time is important in our application and the initial solution is generated by a construction heuristic. Therefore, $T_{start}$ should not be too high. If it is too high, the algorithm goes to a phase called random walk and the probability of accepting a bad solution is high.

The rule mentioned above, for lowering the temperature, defines $T$. The most commonly used one is called exponential cooling. The temperature is set to $T_{start}$ and is adjusted in steps. The temperature adjustment can be expressed mathematically

$$T_{i+1} = \alpha \cdot T_i \tag{7.1}$$

where $\alpha$ is between 0 and 1. The temperature is lowered after a fixed number of iterations, see parameter 7 in table 7.1.

The choice of proper values for $T_{start}$ and $\alpha$ depends highly on problem. In order to find suitable values we experimented with different values for both our algorithms, **SA-A1** and **SA-A2**, see Section 5.4. The results from these experiments can be seen in table 7.2. Same testing values will be used for both algorithms.

Table 7.2: Possible values and test values for the parameters

| Parameter | Possible values | Test values |
|---|---|---|
| Start temperature, $T_{start}$ | $T_{start} \geq 0$ | $\{50, 100, 200\}$ |
| Cooling rate, $\alpha$ | $\alpha \in ]0, 1[$ | $\{0.75, 0.80, 0.85\}$ |

The permutation of the parameters is presented in table 7.3. There are 9 combinations of parameters and each combination has been given a number and will from now on be referred to in figures and in the text by its given number.

Table 7.3: Permutation of parameters for tuning

| Nr. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $\alpha$ | 0.75 | 0.75 | 0.75 | 0.80 | 0.80 | 0.80 | 0.85 | 0.85 | 0.85 |
| T | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 |

## 7.3 Data sets for Tuning

The data sets we have for this problem are based on actual data over 5 days where each day has about 220 orders on average. This number of data sets is not enough to test the algorithms thoroughly so we decided to make 3 data sets out of each day. Firstly, use the whole set of orders, secondly, 160 orders and thirdly, use 100 orders. This gives us 15 data sets instead of 5. Furthermore, to be able to test the algorithms even further we decided to increase the capacity of the trucks up to 50 and test with the same 15 data sets mentioned before. This gives us in total 30 data sets which should allow us to get a better view of the performance of the algorithms. The names of the data sets indicate which day the data is from, the number of orders in the set and the capacity of the trucks used. As an example, `Day2_33` indicates that this data set contains data for the whole set of orders for day 2 and that the capacity of the trucks used is 33. If a whole set of orders is used only information about which day it is and the capacity of the trucks is used. However, if a part of a set of orders for a given day is used as a data set, the number of orders is stated right after the indication of which day it is from. To illustrate this, let us consider data set `Day2_160_50`. It contains 160 orders from Day 2 and the capacity of the trucks used is 50. To sum up, there are data sets with orders from 5 different days, they either contain the whole set of orders, 160 or 100 orders and the capacity of the trucks can either be 33 or 50.

The group of data sets is split in two. One group for the parameter tuning and one for the testing. There are 6 data sets used in the parameter tuning, namely the sets involving data from Day 1. They are the data sets in bold letters in the first line of table 7.4. The table shows the list of the data sets used and their sizes. The sizes of the data sets indicate the number of orders involved.

Table 7.4: Parameter tuning sets and test sets and their sizes

| Name | Size | Name | Size | Name | Size |
|---|---|---|---|---|---|
| **Day1_33, Day1_50** | 247 | **Day1_160_33, Day1_160_50** | 160 | **Day1_100_33, Day1_100_50** | 100 |
| Day2_33, Day2_50 | 214 | Day2_160_33, Day2_160_50 | 160 | Day2_100_33, Day2_100_50 | 100 |
| Day3_33, Day3_50 | 199 | Day3_160_33, Day3_160_50 | 160 | Day3_100_33, Day3_100_50 | 100 |
| Day4_33, Day4_50 | 229 | Day4_160_33, Day4_160_50 | 160 | Day4_100_33, Day4_100_50 | 100 |
| Day5_33, Day5_50 | 212 | Day5_160_33, Day5_160_50 | 160 | Day5_100_33, Day5_100_50 | 100 |

## 7.4 Results

Both algorithms were run 5 times for every permutation and the average performance calculated. Tables with average objective values along with lowest and highest value for every instance can be seen in Appendix C. In order to get a graphical comparison among the data sets, plots are used to illustrate the difference. The lower end of the lines represents the lowest value, the upper end shows the highest value and the knob on the lines shows the average values.

### 7.4.1 Results for algorithm SA-A1

The results from the tuning of the SA-A1 algorithm can be seen in Figs. 7.1 and 7.2. They show the average value and spread for each permutation. The plots reveal that the permutations containing $T_{start} = 50$ give the best results in the majority of the tests. These are permutations 1, 4, and 7. If we take a closer look at these three we see that permutations 1 gives the best results in all cases. We therefore conclude that permutation 1, $T_{start} = 50$ and $\alpha = 0.75$, should be used when testing the **SA-A1** algorithm further.

### 7.4.2 Results for algorithm SA-A2

Plots for all 6 tuning sets and algorithm **SA-A2** can be seen in Figs. 7.3 and 7.4. They show the average values and the spread for each permutation.

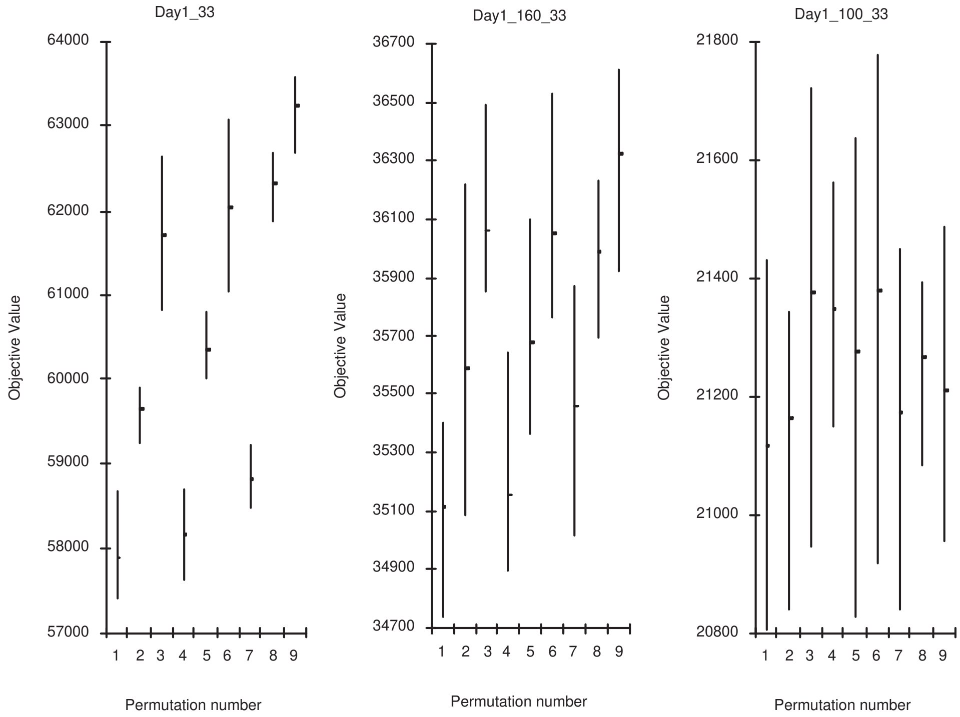


Figure 7.1: Results for algorithm SA-A1 from data sets with truck capacity 33

When studying the results from the tuning in Figs 7.3 and 7.4, it can be seen that permutations 1, 4 and 7 give the best results in the majority of the cases. These three permutations all have the temperature $T_{start} = 50$. If we take a closer look at permutations 1, 4 and 7 in the figures, we see that out of these three, permutation 1 gives over all best results, so the values $T_{start} = 50$ and $\alpha = 0.75$ will be used when we test the algorithm further.

## 7.5 Summary

After tuning both algorithms with the tuning test set we concluded that permutation 1 gives over all best results for both algorithms. As a result of that, we will use the initial temperature $T_{start} = 50$ and cooling rate $\alpha = 0.75$ when testing the algorithms with the test data sets.

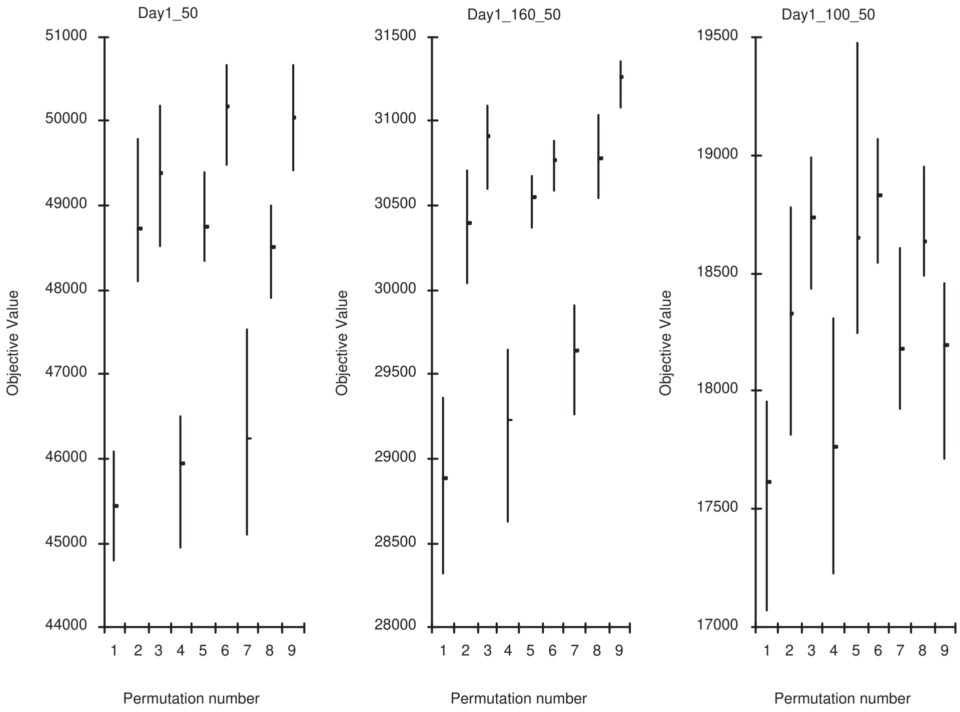


Figure 7.2: Results for algorithm SA-A1 from data sets with truck capacity 50

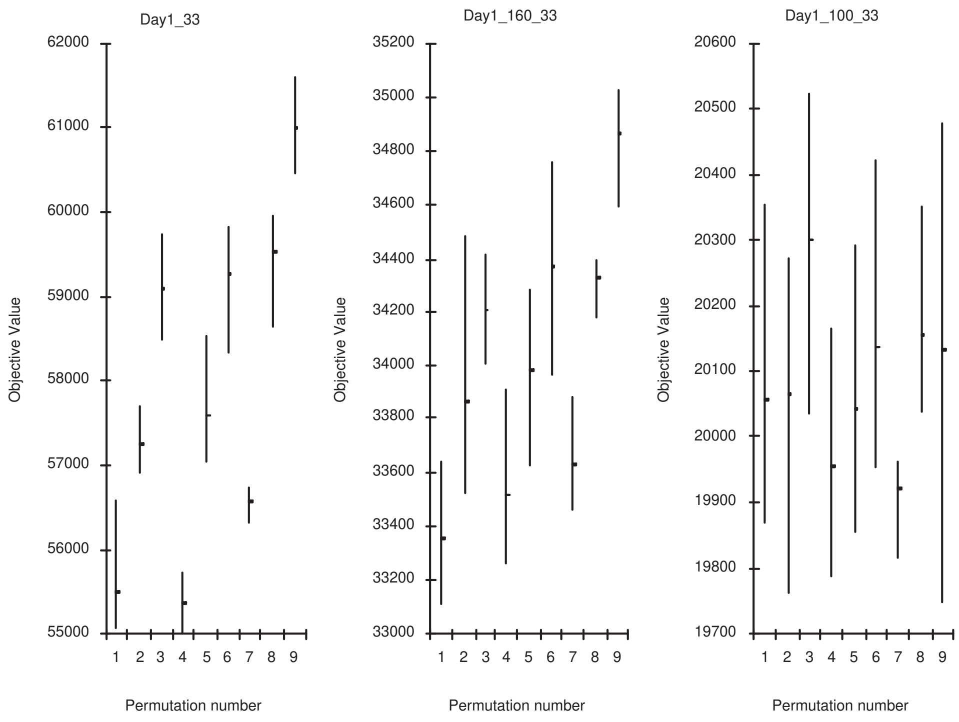


Figure 7.3: Results for algorithm SA-A2 from data sets with truck capacity 33

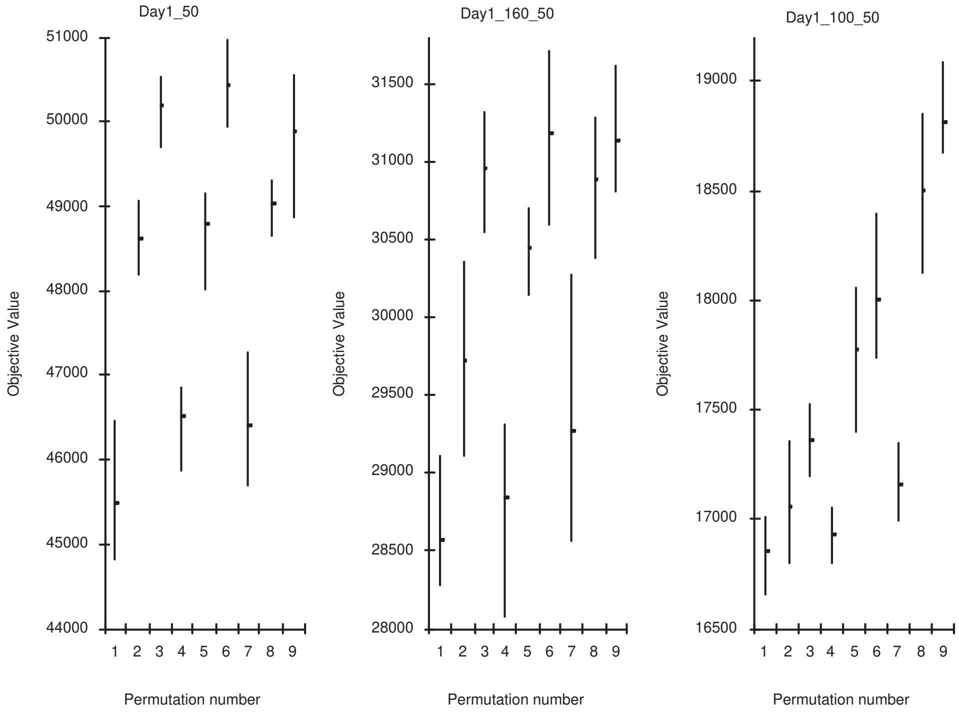


Figure 7.4: Results for algorithm SA-A2 from data sets with truck capacity 50

# Chapter 8

# Experimental Results

We tested both algorithms, **SA-A1** and **SA-A2**. To test the algorithms, they were run 5 times for each of the 24 test data sets. The test sets are listed in non-bold letters in table 7.4. The average objective value was calculated for each data set and presented in tables along with the lowest and highest values for every set.

## 8.1 Comparing the algorithms

Tables 8.1 and 8.2 show the results for data sets with capacity 33, for algorithm **SA-A1** and **SA-A2** respectively. When comparing the values we find that algorithm **SA-A2** gives better results for all instances than **SA-A1**.

Table 8.1: Results for data sets with truck capacity 33 using algorithm **SA-A1**

| Data set | Lowest | Average | Highest | Lower Bound | GAP |
|---|---|---|---|---|---|
| Day2_33 | 50198.3 | 50661.4 | 51036.3 | - | - |
| Day3_33 | 45109.3 | 45766.8 | 47109.8 | - | - |
| Day4_33 | 53211.3 | 53963.7 | 54612.9 | - | - |
| Day5_33 | 48727.9 | 49047.9 | 49413 | - | - |
| Average | | 49860 | | | |
| Day2_160_33 | 37866.3 | 38223.8 | 38683.2 | - | - |
| Day3_160_33 | 36071.8 | 36245.2 | 36470.6 | - | - |
| Day4_160_33 | 37246.6 | 37510.9 | 37959.5 | - | - |
| Day5_160_33 | 36606.1 | 37072 | 37341.4 | - | - |
| Average | | 37263 | | | |
| Day2_100_33 | 23065.4 | 23525.1 | 23826.1 | 7326.8 | 221.1 |
| Day3_100_33 | 22548 | 22724.7 | 22851.6 | 7296.4 | 211.5 |
| Day4_100_33 | 22043.7 | 22246.8 | 22400.1 | 7245.2 | 207.1 |
| Day5_100_33 | 22990.5 | 23400.7 | 23962.6 | 7483.6 | 212.7 |
| Average | | 22974.3 | | | 213.1 |

Table 8.2: Results for data sets with truck capacity 33 using algorithm **SA-A2**

| Data set | Lowest | Average | Highest | Lower Bound | GAP |
|---|---|---|---|---|---|
| Day2_33 | 48249 | 48464.9 | 49107 | - | - |
| Day3_33 | 43939.5 | 44158.3 | 44559 | - | - |
| Day4_33 | 51677.6 | 52006.7 | 52388.5 | - | - |
| Day5_33 | 46991.3 | 47163.5 | 47502.8 | - | - |
| Average | | 47948.4 | | | |
| Day2_160_33 | 35718.7 | 36012.1 | 36284.4 | - | - |
| Day3_160_33 | 34627 | 34872.4 | 35320.1 | - | - |
| Day4_160_33 | 35483.4 | 35611.5 | 35808.3 | - | - |
| Day5_160_33 | 34830.5 | 35129.9 | 35627.4 | - | - |
| Average | | 35406.5 | | | |
| Day2_100_33 | 22203.3 | 22382.9 | 22490.1 | 7326.8 | 205.5 |
| day3_100_33 | 21410.2 | 21562.6 | 21860.3 | 7296.4 | 195.5 |
| Day4_100_33 | 20944.1 | 21233.9 | 21583.1 | 7245.2 | 193.1 |
| Day5_100_33 | 21829.2 | 21980.1 | 22147.9 | 7483.6 | 193.7 |
| Average | | 21789.9 | | | 197.0 |

The results for data sets with capacity 50 are shown in tables 8.3 and 8.4. For data sets with 100 orders we get better results using algorithm **SA-A2**. However, we get very similar results for both algorithms when solving for 160 orders, but a bit better results with **SA-A2** for a whole set of orders.

Table 8.3: Results for data sets with truck capacity 50 using algorithm **SA-A1**

| Data set | Lowest | Average | Highest | Lower Bound | GAP |
|---|---|---|---|---|---|
| Day2_50 | 38988.6 | 39658.3 | 40208.5 | - | - |
| Day3_50 | 35652.5 | 36174.9 | 37194.1 | - | - |
| Day4_50 | 41534.4 | 42411.7 | 43263.8 | - | - |
| Day5_50 | 38562.5 | 39173.3 | 39747.5 | - | - |
| Average | | 39354.6 | | | |
| Day2_160_50 | 30337.9 | 31045.4 | 31458.6 | - | - |
| Day3_160_50 | 29050 | 29355.8 | 29774.8 | - | - |
| Day4_160_50 | 29987.5 | 30472.8 | 30756.4 | - | - |
| Day5_160_50 | 29325 | 30169.1 | 30922.5 | - | - |
| Average | | 30260.8 | | | |
| Day2_100_50 | 19662 | 19902 | 20032 | 7326.8 | 171.6 |
| Day3_100_50 | 18589.6 | 18822.7 | 19085 | 7296.4 | 158.0 |
| Day4_100_50 | 18744.7 | 19185.4 | 20100.6 | 7245.2 | 164.8 |
| Day5_100_50 | 18737.2 | 19285.9 | 19776.2 | 7483.6 | 157.7 |
| Average | | 19299 | | | 163.0 |

These results imply that algorithm **SA-A2** is, as we expected, a better choice. It is not all that surprising since **SA-A1** is in fact not as advanced as **SA-A2**. In algorithm **SA-A1** the

Table 8.4: Results for data sets with truck capacity 50 using algorithm **SA-A2**

| Data set | Lowest | Average | Highest | Lower Bound | GAP |
|---|---|---|---|---|---|
| Day2_50 | 39130.3 | 39506.2 | 39827.5 | - | - |
| Day3_50 | 35389.6 | 35896.4 | 36451.7 | - | - |
| Day4_50 | 41822.4 | 42245.4 | 43036 | - | - |
| Day5_50 | 37455 | 38635.1 | 39544.6 | - | - |
| Average | | 39070.8 | | | |
| Day2_160_50 | 29772.7 | 30722.3 | 31195.6 | - | - |
| Day3_160_50 | 28760.5 | 29410.2 | 30204.8 | - | - |
| Day4_160_50 | 29666.4 | 30547.4 | 31350.8 | - | - |
| Day5_160_50 | 30022.3 | 30393.5 | 30820.8 | - | - |
| Average | | 30268.4 | | | |
| Day2_100_50 | 18751.6 | 19121.8 | 19460.3 | 7326.8 | 161.0 |
| Day3_100_50 | 18113.6 | 18299.6 | 18549.2 | 7296.4 | 150.8 |
| Day4_100_50 | 18066.2 | 18860.5 | 20348.4 | 7245.2 | 160.3 |
| Day5_100_50 | 18064.2 | 18587.3 | 19032.2 | 7483.6 | 148.4 |
| Average | | 18717.3 | | | 155.1 |

number of routes and the number of orders in each route can only change when we apply the *oneInRoute* algorithm, see section 6.1. Algorithm **SA-A2** on the other hand, allows for the possibility of variations in both number of routes and in the number of orders on each route. This advantage alone does in fact make **SA-A2** a much more desirable choice.

## 8.2 The lower bound

Since we do not have any real life solutions to compare our results with, we intended to find a lower bound for the problem by solving our mathematical model with LP relaxation in GAMS. How we did that was discussed in chapter 4. However, we were only able to solve the LP relaxation of the problem for the data sets with 100 orders. Solving data sets with 160 orders or more could not be handled by the CPLEX solver due to the memory requirements. Therefore we did not get any lower bound for the larger data sets.

When solving the LP relaxation of the problem for the data sets with 100 orders in them, it came to our attention that we got the same objective values for the data sets for each day, regardless of which capacity we were using. Nevertheless, the solutions were different.

These lower bounds we found result in a huge gap, as can bee seen in tables 8.1 - 8.4. We believe the reason for that is not that the results from our algorithms are so bad, but that this lower bound is in fact much lower than the actual optimal solution of the problem. To support that we took a small problem, illustrated in section 4.3 and compared its optimal solution to the solution we got using LP relaxation. Here we solved this small problem without using the constraints for multiple use of vehicles. The results can be seen in table 8.5. Looking at those values it is obvious that the LP relaxation gives a much lower objective value than the actual optimal solution.

Table 8.5: Results for different solution methods using a data set with 6 orders, 3 pickup routes and 3 delivery routes.

| | **Solution Method:** | |
|---|---|---|
| **Data set** | **MIP** | **LP relaxation** |
| Data_33 | 1392.34 | 793.84 |
| Data_50 | 1285.79 | 793.84 |

We also solved this very small problem with the **SA-A2** algorithm and we did in fact get the optimal solution for capacity 33. However, it was impossible to solve this instance with capacity 50, since the algorithm is not constructed with such a small problem in mind. When we have only 6 orders the total number of pallets for these 6 orders was only 46 in the data set used. That made it possible to drive all the pickup routes in one route and as a result of that there was no possibility of swapping orders between routes or re-inserting orders.

## 8.3 Experimenting with the running time

Since we do not have a lower bound for all the data sets used and since we do not consider the lower bounds we have to be very reliable, we decided to test the algorithm by running it for a longer period of time. By running the algorithm for a longer period of time we are able to see whether the solution continues to improve, or whether it is already stable. If it is already stable, this is at least the best solution this algorithm can give.

The results we got are shown in figure 8.1 for capacity 33 and in figure 8.2 for capacity 50, where the objective value has been plotted with respect to how long the algorithm has been running. On each graph there are 5 different lines, since we ran the algorithm 5 times for each data set.

These longer runs are only made for the **SA-A2** algorithm and with the data sets for day 2. The original running time was 900 seconds, e.g. 15 minutes, but here we ran the algorithm for 3600 seconds which equals 1 hour.

When looking at the results in figure 8.1 we can see that for the data set with 100 orders we could have run the algorithm for an even shorter time period, since it has stopped improving already after 500 seconds. For the data set with 160 orders the solution stabilizes at 900 seconds and does not improve much after that. When considering the full data set for day 2, the objective is still improving at 900 seconds and continues to improve throughout the running time. Nevertheless it does not improve much after running for 2000 seconds.

The results for longer running time on the data sets with capacity 50 were a bit different, as can be seen in figure 8.2. As could be expected, having more capacity demands longer running time for the algorithm. After running for 900 seconds the objective value for the data set with 100 orders has stopped improving, but the objective value for the data sets with 160 orders and the whole set of orders have still not reached stability. Moreover, the objective value for the data set containing the whole set of orders for the day continues to

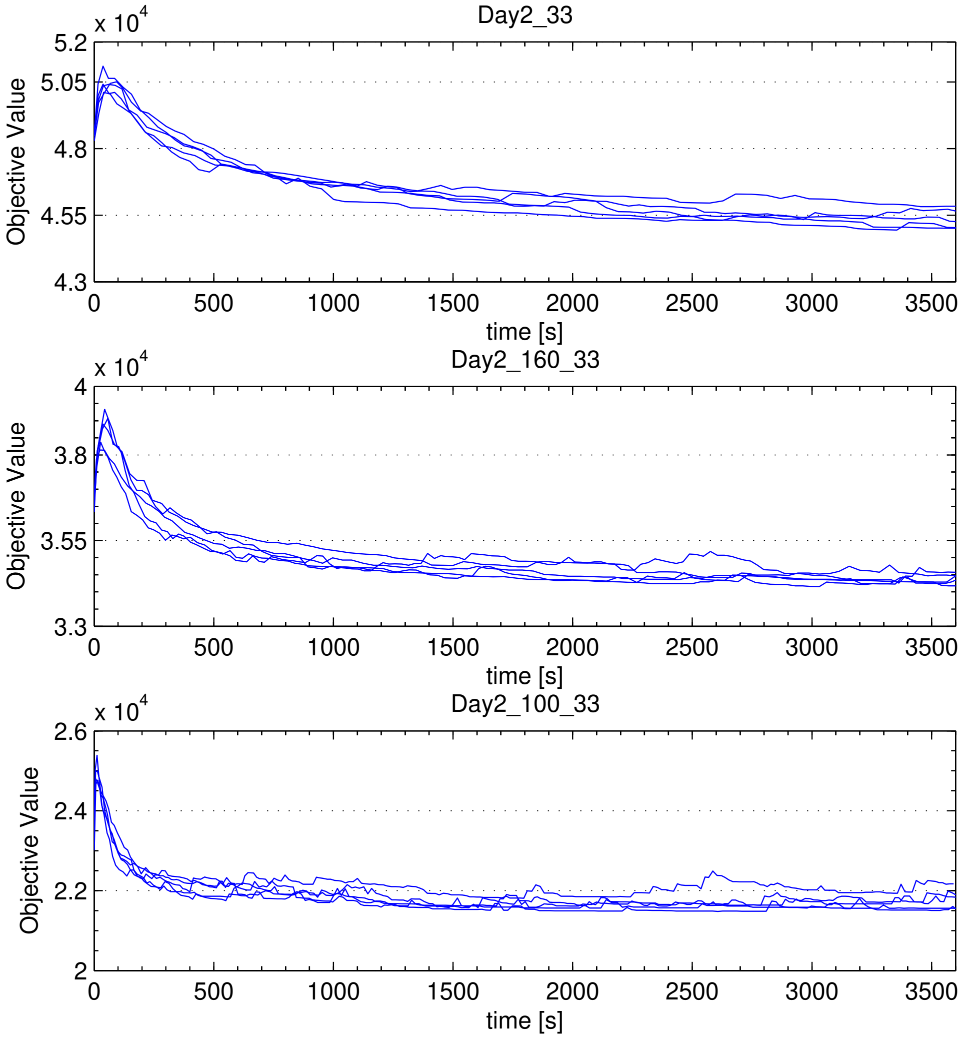


Figure 8.1: The objective values plotted against time for algorithm **SA-A2** using data sets for day 2 with capacity 33. 5 runs for 3600 seconds, i.e. 1 hour.

improve the whole hour we ran it for and the data set for 160 orders is improving for at least 2500 seconds.

When we examined the difference between the results for capacity 33 and capacity 50 we noticed that the temperature decreases more rapidly when we set the capacity to 33 than when it is set to 50. This can be seen in tables D.1 and D.2 in appendix D. In fact, not surprisingly though, the temperature has reached a lot lower value for the data sets that seem to have had long enough running time for their objectives to reach a stable value. This is indeed the nature of SA algorithms.

From this slower temperature decrease for the data sets with capacity 50, we can conclude that our algorithm is slower when the routes are longer, even if we have fewer routes.

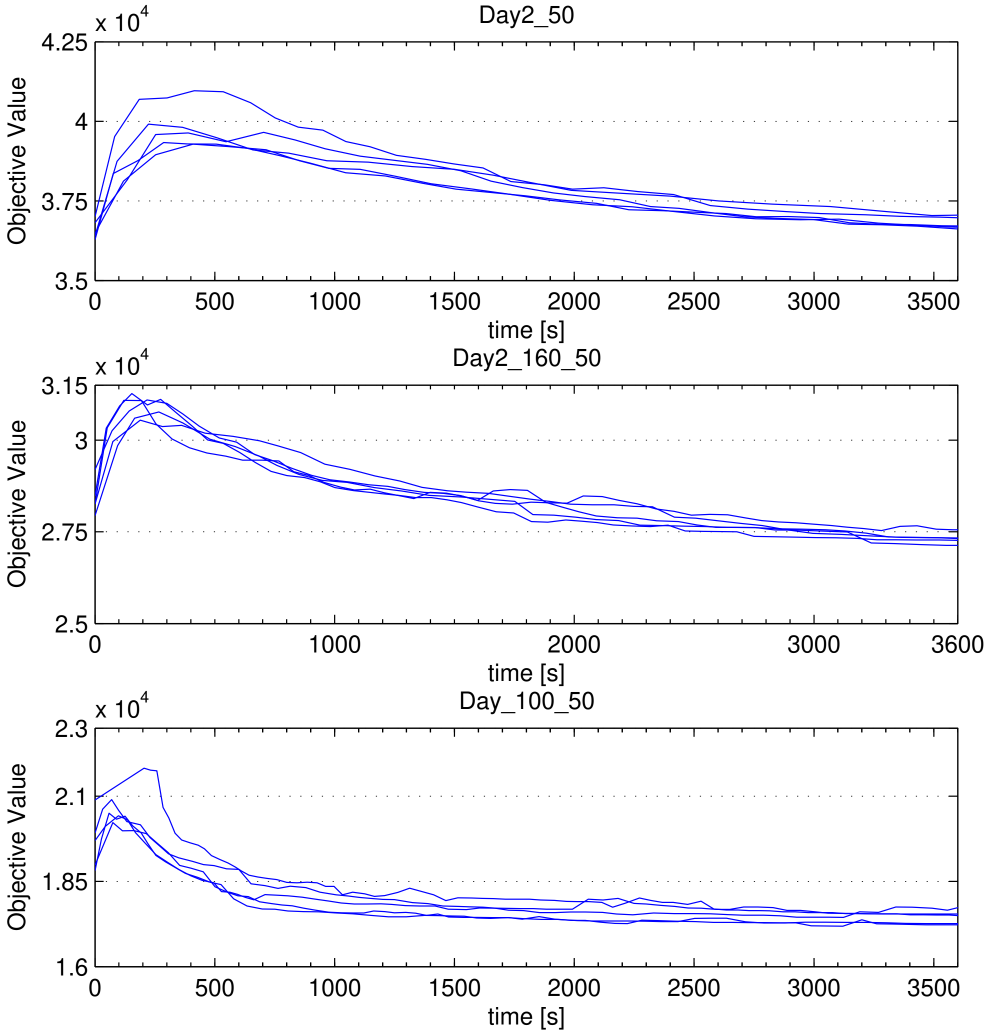


Figure 8.2: The objective values plotted against time for algorithm **SA-A2** using data sets for day 2 with capacity 50. 5 runs for 3600 seconds, i.e. 1 hour.

## 8.4 Using a different comparison

Since the lower bound we intended to use did not prove to be useful, we decided to use the results we got after running the algorithm for a whole hour to compare the results we got out of the original tests with.

When comparing those values, we can assume that if the solutions improve a lot, it indicates that our solutions are not good. If on the other hand, our solutions do not improve much after this longer running time, it suggests that we already have a solution that comes close to being the best solution this algorithm can give. However, we can only hope that it is indeed a good solution of this problem.

The results we got are shown in table 8.6. There we have calculated the average objective values for each data set, for the different running times. We have also calculated the gap between these average objective values to make it easier to compare them. It is calculated in the same way an optimality gap is calculate when we have a lower bound.

These results confirm our believe, that the values for the data sets with capacity 33 are quite good. Especially where we have 100 and 160 orders, but the gaps for those data sets are quite acceptable. For the data sets where the capacity is 50 we get a bigger gap. As said before it seems to be necessary to run the algorithm a bit longer for those data sets, and perhaps also for the data set for the whole day when the capacity is 33.

Table 8.6: Our results, the running time of approximately 15 minutes, compared to the objective values we got after running algorithm **SA-A2** for approximately 1 hour for data sets for day 2.

| Running time | 15 min. | 1 hour | |
|---|---|---|---|
| Data set | Average | Average | GAP |
| Day2_33 | 48464.9 | 45346.9 | 6.9 |
| Day2_160_33 | 36012.1 | 34413.5 | 4.6 |
| Day2_100_33 | 22382.9 | 21791.8 | 2.7 |
| Average | | | 4.7 |
| Day2_50 | 39506.2 | 39827.5 | 7.4 |
| Day2_160_50 | 30722.3 | 31195.6 | 12.5 |
| Day2_100_50 | 19121.8 | 19460.3 | 9.5 |
| Average | | | 9.8 |
| Total Average | | | 7.3 |

## 8.5 Drawing the routes

To get a better idea of how good our solution really is, we decided to plot the routes our results gave. We did that for the 100 order data set, with capacity 33, for day 2. Since the orders are many and the routes also it was not easy to look at a figure with all the routes on it and say whether it was a good solution or not. Therefore we decided to plot only 8 routes on each figure. That way we would at least be able to say whether each route was likely to be sensibly constructed. We plotted the routes for both pickup and delivery and they can be seen in appendix E.

Looking at those figures it is quite obvious that our results are not all that good. Some of the routes look alright, but others are suboptimal. Moreover, some of the routes cross each other and could have been constructed better.

This tells us that our results are probably only suboptimal.

## 8.6 The initial solution

To investigate how important a good initial solution is for the algorithms, we tried running them using a worse initial solution, see 5.5.3. That was done 5 times for each of the data sets with capacity 33. The results can be seen in tables D.3 and D.4 in appendix D.

We decided to plot the results to get a better comparison of how the algorithms performed. The results are shown in figures 8.3 and 8.4 for **SA-A1** and **SA-A2** respectively. The results for each algorithm are shown in three graphs, one for each size of data set and only for capacity 33. On the graphs the data sets with odd numbers are the results for the worse initial solution and the ones with even number for better initial solution, which we used in the parameter tuning and for all tests. So data set number 1 shows the results using the worse initial solution for day 2 and number 2 the results from using the initial solution we have been using all along. Therefore we can compare number 1 and 2, 3 and 4, and so on. The knob on the lines represents the average objective value, the top of the line is the highest objective value and the lower end of the line the lowest objective value.

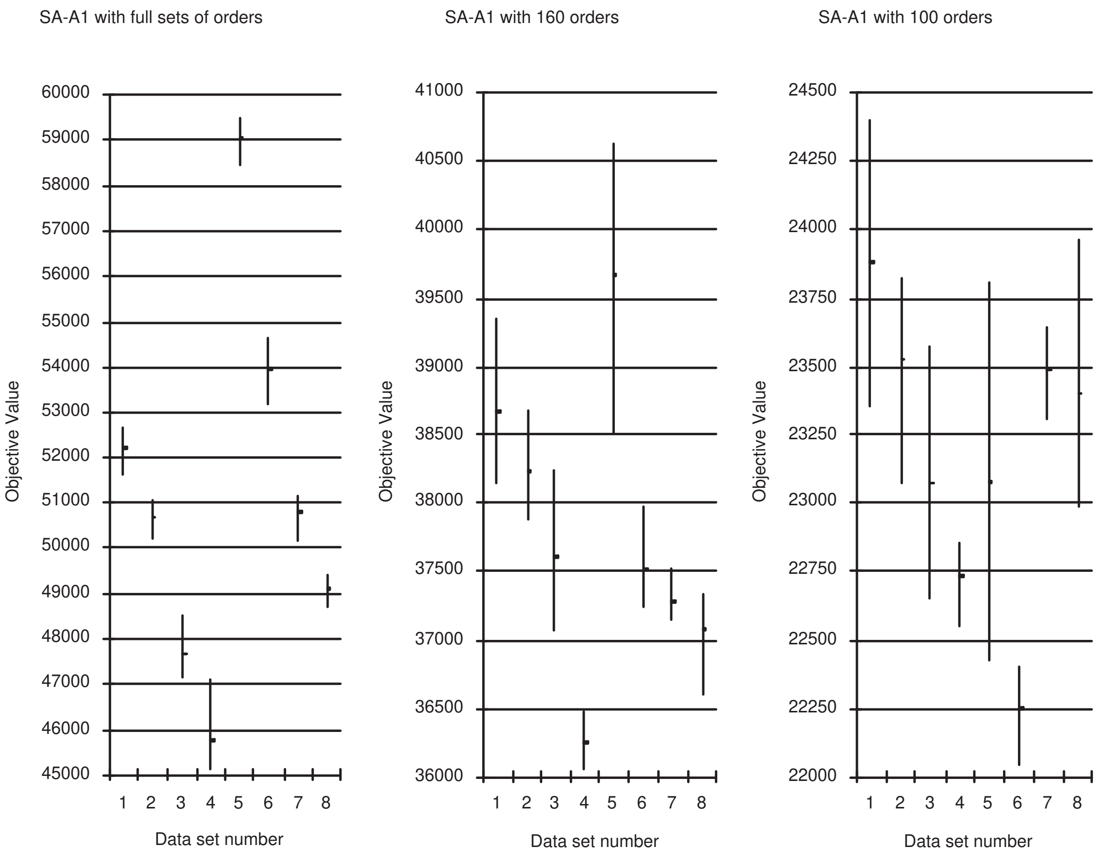


Figure 8.3: Comparing of our results with the results gotten by using a worse initial solution for algorithm **SA-A1**

When looking at figure 8.3 we see that the results for a worse initial solution in algorithm **SA-A1** are definitely worse than the original ones. In some cases they come close, but in many cases far worse. The spread varies, but is overall slightly more for the worse initial solution. Therefore we can conclude that a good initial solution is necessary for this algorithm.

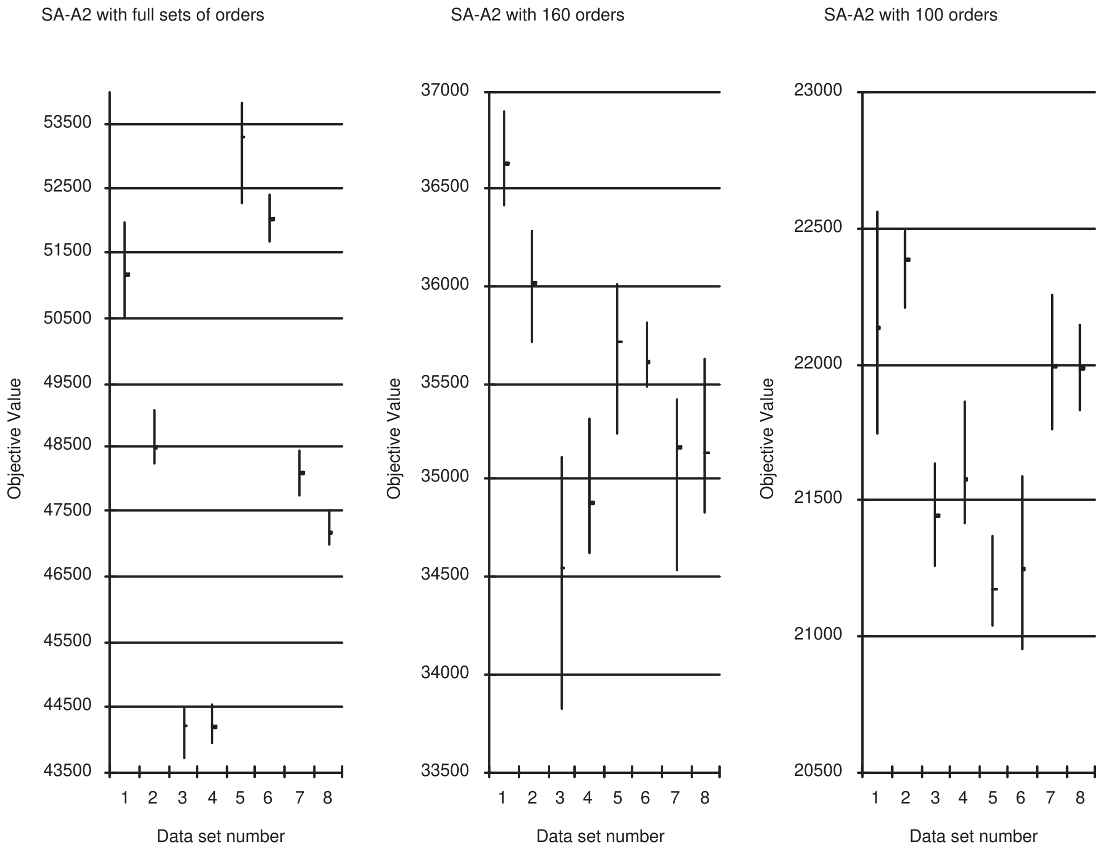


Figure 8.4: Comparison of our results and the results gotten by using a worse initial solution for algorithm **SA-A2**

The comparison of initial solutions for algorithm **SA-A2** in figure 8.4 looks different. When viewing the graph for the data sets with 100 orders, we see that the results with worse initial solution are either equally good or better than the original results. In the graphs for 160 orders it is hard to say which initial solution gives better result, but in the graph for the whole set of orders the worse initial solution seems to give worse objective value. Based on this we can say that a good initial solution is not as important as it is for **SA-A1**, nevertheless it gives better results for larger data sets.

The reason for that is, when solving for larger data sets, it takes longer time to improve the initial solution than for the smaller ones. Therefore we can expect a good initial solution to give a better solution than a bad one when we run the algorithm for a shorter time period. However, we can expect to get similar results for both initial solution if we run the algorithms for a longer period of time.

## 8.7 Summary

The results from testing show that algorithm **SA-A2** has a better performance than algorithm **SA-A1**. The reason for that is that algorithm **SA-A2** is more advanced and its advantages are obviously working.

The LP lower bonds we found did not prove to be usable. We have shown how far below optimality they turned out to be for a very small data set, so there is no reason to think they are any good to compare the results of our algorithms with.

We experimented with longer running times for the algorithms. That gave us a better idea of how good our results from testing were. Looking at how the objective value evolves implied that in some cases the algorithm should have been run a bit longer. Specially for the largest data set and when the vehicles have more capacity, since the temperature decreases more slowly in those cases.

We also decided to take the objective values we got after running the algorithm for an whole hour and use them to compare the results we got in the testing with. We examined the gap and it turned out to be quite low for the smaller data sets with capacity 33. Nevertheless, this confirms that longer running time is needed for the data sets with larger capacity, since the gap is larger in those cases.

The routes for one data set were plotted. We plotted 8 routes on each figure to see more clearly how they were constructed. Looking at those plots it is obvious that our results are probably only suboptimal.

Checking out the importance of the quality of our initial solution gave us some interesting results. We found that the quality of the initial solution is very important for algorithm **SA-A1**. That was like we expected, since that algorithm is not very well suited to improve a bad solution. For algorithm **SA-A2** the initial solution did not seem to be all that important. However, the better initial solution gave slightly better results for the largest data set, namely the data set for whole sets of orders.

# Chapter 9

# Summary and conclusions

## 9.1 Summary

We have analyzed the problem and built a mathematical model of it. The mathematical model is based on the VRP, or in fact linking two VRPs together at a cross docking station. The mathematical model has been proven to be correct and has been implemented in GAMS. The GAMS code can be used to solve a small version of the problem, but it is very time consuming to solve the problem to optimality, even with a very small problem. Since the model we built is based on a model of the VRP, which is NP hard, we know that our problem is also NP hard. Therefore it is not possible to solve the problem to optimality with exact methods. Because of that we decided to use metaheuristics to solve the problem.

We choose to use an SA algorithm to solve the problem and we implemented two different algorithms, one of them being an improvement of the other. The first one, **SA-A1**, is based on a swapping method and the second one, **SA-A2**, is based on a re-insert method in addition to the swapping method. Both algorithms were improved by applying two improvement algorithms, the 2-opt algorithm and an algorithm that attempts to eliminate routes that only transport one order.

When the algorithms had been fully implemented, their most important parameters were tuned and the rest otherwise determined. In the tuning process we used 6 different data sets for each algorithm and tried 3 different values for each of the parameters tuned. When we had selected the best values for the parameters tuned, those values were used in the algorithms when we tested them.

In the testing process, the algorithms were run for 24 different data sets each. In both parameter tuning and testing we ran the algorithms 5 times for each data set, each time for approximately 15 minutes. The results form the tests were then used to compare the performance of the algorithms. This comparison confirmed what we had believed, or that the algorithm **SA-A2** was a better choice. To get a lower bound we solved the mathematical model in GAMS with LP relaxation. However, it was only possible to solve the LP relaxation for the smallest data sets used in the testing, we could not solve for the other two sizes of data sets with the CPLEX solver due to memory requirements. Because of that, and since we did not consider the lower bounds we got to be very reliable, we examined the performance of the algorithms by running them for a longer time period.

We investigated how the objective values were evolving with respect to the running time, and we also used the objective values these longer runs resulted in, to compare our test results with. By doing that we found that for algorithm **SA-A2** the solutions do not improve all that much when we are using the data sets with capacity 33, but when we increase the capacity to 50 it is apparently necessary to run the algorithm for a longer time period.

Finally we considered the importance of a good initial solution for the algorithms and discovered that, as we suspected, a good initial solution was necessary for algorithm **SA-A1**. For algorithm **SA-A2** on the other hand, a good initial solution was not crucial, however, especially for the largest data sets we can expect that worse initial solutions require longer running time of the algorithm.

## 9.2 Conclusions

As mentioned above, we have concluded that this problem is indeed NP hard and can therefore not be solved to optimality for large data sets. However, there are reasons to think it is possible to achieve a relatively good solutions by applying metaheuristics.

We have, by using SA, managed to solve the problem with the large data sets provided. Since we did not have any real life results to compare our results with, we have not been able to fully prove the performance of our algorithm. By looking at the evolution of the objective value with the running time of the algorithm, we have though at least been able to show that we are coming close to getting the best solutions this algorithm can give, when using the algorithm **SA-A2** with data sets with capacity 33. We can also conclude that for data sets with capacity 50 it is reasonable to think that we could, at least for the smallest data sets, come close to the best objective value achievable by the algorithm if we increased the running time a little bit.

However, judging from the plots we made of our routes, our algorithm gives a suboptimal solution. The routes could in many cases have been constructed better and are probably only suboptimal.

# Chapter 10

# Further work

This chapter provides a few ideas for further work or if improvements of our solution methods.

In our mathematical model, we omit the possibility described in the problem description of making it possible not to reload all orders on the cross docking station. It would be worth while to solve this problem with that possibility, since the reloading of orders is both time consuming and includes some expenses. Those expenses could in this way possibly be avoided, given that it would not result in more expensive routes. Adding this feature to our solution methods would likely be rather complicated though.

Many further improvements of the algorithm are possible. The main thing our **SA-A2** algorithm lacks, is the possibility that the number of routes could increase. One way of adding that possibility would be to divide very long routes and make two routes out of them. This would be useful for example if we start out with a initial solution that has very few routes. A solution like that could not be improved by the algorithm without adding the possibility of increasing the number of routes.

Since this algorithm only gives a suboptimal solution at this stage, it would probably be a good idea to try implementing the method above as another neighborhood search for the algorithm. Evolving the algorithm further that way might improve the routes by splitting up the longest ones.

Another thing that we have not done, but should be easy to add to the algorithm, is to implement multiple use of vehicles, as is done in the mathematical model.

# Bibliography

[1] T. Duncan. Neighbourhood search and the vehicle routing problem. Technical report, Artificial Intelligence Application Institute, University of Edinburgh, 1995.

[2] M. L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42:626–642, 1994.

[3] Sonia Hajri-Gabouj and Saber Darmoul. A hybrid evolutionary approach for a vehicle routing problem with double time windows for the depot and multiple use of vehicles. *Studies in Informatics and Control*, 12(4):253–268, 2003.

[4] V. Jayaraman and A. Ross. A simulated annealing methodology to distribution network design and management. *European Journal of Operational Research*, 144(3):629–645, 2003.

[5] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. A preliminary version of a chapter that appeared in the book Local Search in Combinatorial Optimization, E.H.L.Aarts and J.K.Lenstra (eds.), John Wiley and Sons, 1997, pp. 215-310., November 1995.

[6] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.

[7] Paolo Toth and Daniele Vigo, editors. *The vehicle routing problem.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

# Appendix A

# Mathematical model

Parameters

$$i - \text{order}, \qquad i = 0, 1, \ldots, N+1 \tag{A.1}$$

$$j - \text{order}, \qquad j = 0, 1, \ldots, N+1 \tag{A.2}$$

$$k - \text{pickup routes}, \qquad k = 1, \ldots, K \tag{A.3}$$

$$l - \text{delivery routes}, \qquad l = 1, \ldots, L \tag{A.4}$$

$$k' - \text{pickup route following route } k, \qquad k' = 1, \ldots, K \tag{A.5}$$

$$l' - \text{delivery route following route } l, \qquad l' = 1, \ldots, L \tag{A.6}$$

Constants

$$A_{ij} - \text{travelling time between pick-ups} \tag{A.7}$$

$$B_{ij} - \text{travelling time between deliveries} \tag{A.8}$$

$$q_i - \text{size of order } i \tag{A.9}$$

$$S - \text{time needed to serve vehicle on each stop} \tag{A.10}$$

$$P - \text{time needed to load/unload one pallet} \tag{A.11}$$

$$ck - \text{truck capacity on pick-up routes } ck \tag{A.12}$$

$$cl - \text{truck capacity on delivery routes } cl \tag{A.13}$$

$$R_{max} - \text{Maximum working time for vehicles on delivery routes} \tag{A.14}$$

$$T_{max} - \text{Maximum working time for vehicles on pick-up routes} \tag{A.15}$$

$$\epsilon - \text{infinitesimal positive number} \tag{A.16}$$

Service time at customer $i$ for both pick-up and delivery:

$$L_i = S + q_i \cdot P \qquad \forall\ i = 1, \ldots, N. \tag{A.17}$$

Variables

$$x_{ijk} = \begin{cases} 1 & \text{if order } j \text{ is picked up directly after order } i \text{ on route } k \\ 0 & \text{otherwise} \end{cases} \tag{A.18}$$

$$z_{ijl} = \begin{cases} 1 & \text{if order } j \text{ is delivered directly after order } i \text{ on route } l \\ 0 & \text{otherwise} \end{cases} \tag{A.19}$$

$$y_{ik} = \begin{cases} 1 & \text{if order } i \text{ is picked up on route } k \quad \forall\, i = 1, \dots N \\ 0 & \text{otherwise} \end{cases} \tag{A.20}$$

$$v_{il} = \begin{cases} 1 & \text{if order } i \text{ is delivered on route } l \quad \forall\, i = 1, \dots N \\ 0 & \text{otherwise} \end{cases} \tag{A.21}$$

$$\alpha_{kk'} = \begin{cases} 1 & \text{if route } k \text{ followed by route } k' \text{ are both} \\ & \quad \text{assigned to the same vehicle} \\ 0 & \text{otherwise} \end{cases} \tag{A.22}$$

$$\beta_{ll'} = \begin{cases} 1 & \text{if route } l \text{ followed by route } l' \text{ are both} \\ & \quad \text{assigned to the same vehicle} \\ 0 & \text{otherwise} \end{cases} \tag{A.23}$$

$D -$ Number of driven routes in pickup (A.24)

$E -$ Number of driven routes in delivery (A.25)

$TL_k -$ total time to unload truck on route $k$ at depot (A.26)

$RL_l -$ total time to load truck on route $l$ at depot (A.27)

$TotSerTime -$ the Total service time is a constant based on $L_i$, $TL_k$ and $RL_l$ (A.28)

$t_{ik} -$ the time label for when the vehicle on route $k$ has been loaded and is ready to leave $i$ (A.29)

$r_{il} -$ the time label for when the vehicle on route $l$ has been unloaded and is ready to leave $i$ (A.30)

$T_k -$ the finishing time at depot of pickup route $k$ (A.31)

$R_l -$ the starting time at depot of delivery route $l$ (A.32)

$a -$ number of required vehicles on pickup routes (A.33)

$b -$ number of required vehicles on delivery routes (A.34)

Objective Function

$$\min \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} \left( A_{ij} \sum_{k=1}^{K} x_{ijk} + B_{ij} \sum_{l=1}^{L} z_{ijl} \right) + TotSerTime \tag{A.35}$$

Subject To

Constraints that guaranty that if a particular route leaves a given location, the order to/from that location is picked up on that route:

$$\sum_{j=1}^{N} x_{ijk} = y_{ik}, \quad \forall\ i = 1, \ldots, N,\ k = 1, \ldots, K \tag{A.36}$$

$$\sum_{j=1}^{N} z_{ijl} = v_{il}, \quad \forall\ i = 1, \ldots, N,\ l = 1, \ldots, L \tag{A.37}$$

Constraints to make sure that every order is picked up/delivered once and only once:

$$\sum_{k=1}^{K} y_{ik} = 1, \quad \forall\ i = 1, \ldots, N \tag{A.38}$$

$$\sum_{l=1}^{L} v_{il} = 1, \quad \forall\ i = 1, \ldots, N \tag{A.39}$$

Flow constraints:

$$\sum_{j=0}^{N+1} x_{ijk} = \sum_{j=0}^{N+1} x_{jik}, \quad \forall\ i = 1, \ldots, N,\ k = 1, \ldots, K \tag{A.40}$$

$$\sum_{j=0}^{N+1} z_{ijl} = \sum_{j=0}^{N+1} z_{jil}, \quad \forall\ i = 1, \ldots, N,\ l = 1, \ldots, L \tag{A.41}$$

Constraints to guaranty that every route leaves the depot and returns to it again:

$$\sum_{j=0}^{N+1} x_{0,j,k} = \sum_{i=0}^{N+1} x_{i,(N+1),k} = 1, \quad \forall\ k = 1, \ldots, K \tag{A.42}$$

$$\sum_{j=0}^{N+1} z_{0,j,l} = \sum_{i=0}^{N+1} z_{i,(N+1),l} = 1, \quad \forall\, l = 1, \ldots, L \tag{A.43}$$

Constraints to guaranty that routes do not return to the starting depot nor leave the return depot:

$$\sum_{j=0}^{N+1} x_{(N+1),j,k} = \sum_{i=0}^{N+1} x_{i,0,k} = 0, \quad \forall\, k = 1, \ldots, K \tag{A.44}$$

$$\sum_{j=0}^{N+1} z_{(N+1),j,l} = \sum_{i=0}^{N+1} x_{i,0,l} = 0, \quad \forall\, l = 1, \ldots, L \tag{A.45}$$

Capacity constraints:

$$\sum_{i=1}^{N} q_i y_{ik} \leq ck, \quad \forall\, k = 1, \ldots, K \tag{A.46}$$

$$\sum_{i=1}^{N} q_i v_{il} \leq cl, \quad \forall\, l = 1, \ldots, L \tag{A.47}$$

Constraints to make sure every routes leaves the depot (if not driven they go from 0 to $N+1$):

$$\sum_{j=1}^{N+1} \sum_{k=1}^{K} x_{0jk} = K \tag{A.48}$$

$$\sum_{j=1}^{N+1} \sum_{l=1}^{L} z_{0jl} = L \tag{A.49}$$

Constraints that define the number of routes driven:

$$\sum_{j=1}^{N} \sum_{k=1}^{K} x_{0jk} = D \tag{A.50}$$

$$\sum_{j=1}^{N} \sum_{l=1}^{L} z_{0jl} = E \tag{A.51}$$

Total time to unload truck on route $k$ at depot:

$$TL_k = \sum_{j=1}^{N} (y_{ik} \cdot q_i) \cdot P + S * (1 - x_{0(N+1)k}), \quad \forall\, k = 1, \ldots, K \tag{A.52}$$

Total time to load truck on route $l$ at depot:

$$RL_l = \sum_{j=1}^{N} (v_{il} \cdot q_i) \cdot P + S * (1 - z_{0(N+1)l}), \quad \forall\, l = 1, \ldots, L \tag{A.53}$$

The Total service time is a constant based on 3.18, 3.19 and 3.20:

$$TotSerTime = 2 \cdot \sum_{j=1}^{N} L_i + \sum_{k=1}^{K} TL_k + \sum_{l=1}^{L} RL_l \tag{A.54}$$

Time constraints for pick-up routes:

$$x_{ijk}(t_{ik} + A_{ij} + L_j - t_{jk}) \leq 0, \quad \forall\, i = 0, ..., N;\ j = 1, ..., N;\ k = 1, ..., K \tag{A.55}$$

$$x_{i(N+1)k}(t_{ik} + A_{i(N+1)} + TL_k - T_k) \leq 0, \qquad \forall\, i = 1, ..., N;\ k = 1, ..., K \tag{A.56}$$

Time constraints for delivery routes:

$$z_{0jl}(R_l + RL_l + B_{0j} + L_j - r_{jl}) \leq 0, \qquad \forall\, j = 1, ..., N; \quad l = 1, ..., L \tag{A.57}$$

$$z_{ijl}(r_{il} + B_{ij} + L_j - r_{jl}) \leq 0, \quad \forall\, i = 1, ..., N; \quad j = 1, ..., N; \quad l = 1, ..., L \tag{A.58}$$

$$z_{i(N+1)l}(r_{il} + B_{i(N+1)} - r_{(N+1)l}) \leq 0, \qquad \forall\, i = 1, ..., N; \quad l = 1, ..., L \tag{A.59}$$

Constraints on the finishing time of the delivery routes:

$$r_{(N+1)l} \leq R_{max}, \qquad \forall\, l = 1, ..., L \tag{A.60}$$

Constraints to keep the time labels zero for routes not driven and locations not visited on the route:

$$t_{ik} \leq M \cdot y_{ik} \quad \forall\ i = 1, ..., N; \quad k = 1, ..., K \tag{A.61}$$

$$r_{il} \leq M \cdot v_{il} \quad \forall\ i = 1, ..., N; \quad l = 1, ..., L \tag{A.62}$$

Special constraints for the final time label:

$$r_{(N+1)l} \leq M \cdot (1 - z_{0(N+1)l}) \quad \forall\ l = 1, ..., L \tag{A.63}$$

Constraints to keep the finishing times of pickup routes zero for routes not driven:

$$T_k \leq M \cdot (1 - x_{0(N+1)k}) \quad \forall\ k = 1, ..., K \tag{A.64}$$

Constraints to keep the starting times of delivery routes zero for routes not driven:

$$R_l \leq M \cdot (1 - z_{0(N+1)l}) \quad \forall\ l = 1, ..., L \tag{A.65}$$

Constraints to make sure that the delivery route does not start until the pick-up route with the orders involved has ended (pickup guaranty):

$$T_k \leq R_l + M(-y_{ik} - v_{il} + 2) \quad \forall\ i = 1, ..., N; \quad k = 1, ..., K; \quad l = 1, ..., L \tag{A.66}$$

Multi-use-of-vehicles constraints for pick-up routes:

$$\alpha_{kk'} \leq 1 + \frac{t_{0k'} - T_k}{T_{max}} - x0(N+1)k, \quad \forall\ k = 1, ..., K;\ k' = 1, ..., K;\ k \neq k' \tag{A.67}$$

Constraints to make sure there is no more than one route $k'$ following route $k$ directly:

$$\sum_{k'=1}^{K} \alpha_{kk'} \leq 1, \quad \forall\ k = 1, ..., K \tag{A.68}$$

Constraints that guaranty the vehicle driving route $l'$ does not drive more than one route $l$ directly before driving route $l'$:

$$\sum_{k=1}^{K} \alpha_{kk'} \leq 1, \quad \forall\ k' = 1, ..., K \tag{A.69}$$

Number of required vehicles on pickup routes:

$$a = D - \sum_{k=1}^{K} \sum_{k'=1}^{K} \alpha_{kk'} \tag{A.70}$$

Multi-use-of-vehicles constraints for delivery routes:

$$\beta_{ll'} \leq 1 + \frac{R_{l'} - r_{(N+1)l}}{R_{max}} - z_{0(N+1)l}, \quad \forall\ l = 1, ..., L;\ l' = 1, ..., L;\ l \neq l' \tag{A.71}$$

Constraints to make sure there is at most one route $l'$ following route $l$ directly on the same vehicle:

$$\sum_{l'=1}^{L} \beta_{ll'} \leq 1, \quad \forall\ l = 1, ..., L \tag{A.72}$$

Constraints that guaranty the vehicle driving route $l'$ does not drive more than one route $l$ directly before driving route $l'$:

$$\sum_{l=1}^{L} \beta_{ll'} \leq 1, \quad \forall\ l' = 1, ..., L \tag{A.73}$$

Number of required vehicles on delivery routes:

$$b = E - \sum_{l=1}^{L} \sum_{l'=1}^{L} \beta_{ll'} \tag{A.74}$$

# Appendix B

# GAMS source code

```
$Title  A Routing and Distribution Problem

$eolcom //

option iterlim=999999999;   // avoid limit on iterations option
reslim=900000;              // timelimit for solver option
optcr=0.0;                  // gap tolerance option solprint=OFF;
// include solution print in .lst file option limrow=500;
// limit number of rows in .lst file option limcol=500;
// limit number of columns in .lst file
//------------------------------------------------------------------

  Sets

   i   Orders

      / 0*7 /


   k   Pick-up Routes

      / 1*3 /

   l   Delivery Routes

      / 1*3 /;

   alias (i,j);
   alias (k,km);
   alias (l,lm);


  Parameter ELL(i)  Service time at customer i for both pickup and delivery


/0  0
1   26
2   20
3   14
4   14
5   13
6   19
7   0
/;


  Parameter q(i)  size of order i
```

```
/0  0
1   16
2   10
3   4
4   4
5   3
6   9
7   0
/;



 Table A(i,j) Travelling time between pick-ups

    0   1   2   3   4   5   6   7
0   1000    49.9    67.7    37.8    51.8    73.4    53.3    0
1   49.9    1000    33.7    12.2    62.7    33.9    3.5 49.9
2   67.7    33.7    1000    40.1    50.5    8.4 32.3    67.7
3   37.8    12.2    40.1    1000    57.6    42.5    15.7    37.8
4   51.8    62.7    50.5    57.6    1000    58.9    64.4    51.8
5   73.4    33.9    8.4 42.5    58.9    1000    31.8    73.4
6   53.3    3.5 32.3    15.7    64.4    31.8    1000    53.3
7   0   49.9    67.7    37.8    51.8    73.4    53.3    1000;



 Table B(i,j) Travelling time between deliverys


    0   1   2   3   4   5   6   7
0   1000    153.58  90.32   149.01  214.27  93.38   124.06  0
1   153.58  1000    72.74   66.72   187.18  71.41   97.29   71.45
2   90.32   72.74   1000    60.82   157.50  3.39    52.22   65.47
3   149.01  66.72   60.82   1000    120.68  57.45   41.16   110.01
4   214.27  187.18  157.50  120.68  1000    154.80  105.31  221.26
5   93.38   71.41   3.39    57.45   154.80  1000    49.49   67.59
6   124.06  97.29   52.22   41.16   105.31  49.49   1000    116.40
7   0   71.45   65.47   110.01  221.26  67.59   116.40  1000;




  Scalar ck  Truck capacity on pickup routes
    / 33 /;

  Scalar cl  Truck capacity on delivery routes
    / 33 /;

  Scalar Rmax Maximum working time for the vehicles
    / 960 /;

  Scalar Tmax Maximum working time for the vehicles
    / 960 /;

  Scalar M large positive number
    / 10000 /;

  Scalar Epsilon infinitesimal positive number
    / 0.001 /;



  Positive Variables

   TL(k)           Total time to unload trucks on route k at depot
   RL(l)           Total time to load trucks on route l at depot
   t_loaded(i,k)        Time label for when the vehicle on route
k has been loaded an is ready to leave i
   r_unloaded(i,l)      Time label for when the vehicle on route l has
```

```
been unloaded and is ready to leave i
   T(k)                 The finishing time at depot of pick-up route k
   R(l)                 The starting time at depot of delivery route l
   TotSerTime          The total service time
   a_vehicles          Number of vehicles used in pickup
   b_vehicles          Number of vehicles used in delivery;


  Free Variables

   Obj                 Total time in hours;


  Binary Variables

   x(i,j,k)   Shows weather order j is picked up directly after
order i on route k
   z(i,j,l)   Shows weather order j is delivered directly after
order i on route l
   y(i,k)     Shows if order i is picked up on route k
   v(i,l)     Shows if order i is delivered on route l
   alfa(k,km)  Shows if route k is followed by route k
   beta(l,lm)  Shows if route l is followed by route l;


  Equations

    MinimumTime         Define objective function

    GoFromDepot(k)      All vehicles on route k leave out-depot
    GoFromDepotDel(l)   All vehicles on route l leave out-depot

    NoLeavingDepotAgain(k)   No vehicle on route k leaves in-depot
    NoLeavingDepotAgainDel(l)  No vehicle on route l leaves in-depot

    FlowConstraint(i,k)     Flow constraints for pickup
    FlowConstraintDel(i,l)  Flow constraints for delivery

    GoToDepot(k)        All vehicles on route k return to in-depot
    GoToDepotDel(l)     All vehicles on route l return to in-depot

    NoGoFromFromdepot(k)    No vehicle on route k returns to out-depot
    NoGoFromFromdepotDel(l) No vehicle on route l returns to out-depot


    TotalUnloadTime(k)        Total time to unload trucks on route k at
depot
    TotalLoadTime(l)          Total time to load trucks on route l at depot


    PickupLeaveOnce(i,k)      Makes sure truck on route k leaves each
customer only once
    DeliveryLeaveOnce(i,l)    Makes sure truck on route l leaves each
customer only once

    PickupCapacity(k)         Truck capacity on route k
    DeliveryCapacity(l)       Truck capacity on route l

    PickupOnce(i)            Makes sure every order is picked up once
and only once
    DeliveryOnce(i)          Makes sure every order is delivered once
and only once

    NoYatStartDepot      Makes sure y is zero at starting depot
    NoYatFinDepot        Makes sure y is zero at finishing depot

    NoVatStartDepot      Makes sure v is zero at starting depot
    NoVatFinDepot        Makes sure v is zero at finishing depot
```

```
    TimeConstrPickup1(i,j,k)  Time constraint for pickup routes
    TimeConstrPickup2(i,k)    Time constraint for pickup routes

    TimeConstrDelivery1(j,l)   Time constraint for delivery routes
    TimeConstrDelivery2(i,j,l)   Time constraint for delivery routes
    TimeConstrDelivery3(i,l)     Time constraint for delivery routes
    TimeConstrDelivery4(l)   Time constraint for delivery routes

    DeliveryPickup(i,k,l)     Delivery route does not start untill

  Tconstr(k)
  Rconstr(l)

  NoTime(i,k)
  NoTimeDel(i,l)

  runloadFinal(l)

    AlfaSama(k,km)
    BetaSama(l,lm)

    MultiUsePickup(k,km)
    MultiUseDelivery(l,lm)

    AlfaConstr(k)        Route can only be followed directly by one
other route at most
    AlfaConstr2(km)

    BetaConstr(l)
    BetaConstr2(lm)

    NumberOfVehicles
    NumberOfVehiclesDel

    TotSerTimeCalculated     Total service time
;




  MinimumTime..        Obj  =e=  sum((i,j), A(i,j) * sum(k, x(i,j,k)))
+ sum((i,j), B(i,j) * sum(l, z(i,j,l)))
+ TotSerTime + a_vehicles * 10 + b_vehicles * 10;


  GoFromDepot(k)..      sum(j, x("0",j,k)) =e= 1;
  GoFromDepotDel(l)..       sum(j, z("0",j,l)) =e= 1;

  NoLeavingDepotAgain(k)..   sum(j, x("7",j,k)) =e= 0;
  NoLeavingDepotAgainDel(l)..   sum(j, z("7",j,l)) =e= 0;


  FlowConstraint(i,k)$(ord(i)<>1 and ord(i)<>8)..
sum(j, x(i,j,k)) =e= sum(j, x(j,i,k));
  FlowConstraintDel(i,l)$(ord(i)<>1 and ord(i)<>8)..
sum(j, z(i,j,l)) =e= sum(j, z(j,i,l));


  GoToDepot(k)..        sum(i, x(i,"7",k)) =e= 1;
  GoToDepotDel(l)..     sum(i, z(i,"7",l)) =e= 1;

  NoGoFromFromdepot(k)..   sum(i, x(i,"0",k)) =e= 0;
  NoGoFromFromdepotDel(l)..   sum(i, z(i,"0",l)) =e= 0;


  TotalUnloadTime(k) ..     TL(k) =e= sum(i, (y(i,k)*q(i)) * 1)
+ 10 * (1 - x("0","7",k));
  TotalLoadTime(l) ..       RL(l) =e= sum(i, (v(i,l)*q(i)) * 1)
+ 10 * (1 - z("0","7",l));
```

```
  PickupLeaveOnce(i,k)$(ord(i)<>1 and ord(i)<>8) ..
sum(j, x(i,j,k)) =e= y(i,k);
  DeliveryLeaveOnce(i,l)$(ord(i)<>1 and ord(i)<>8) ..
sum(j, z(i,j,l)) =e= v(i,l);


  PickupCapacity(k)..       sum(i, q(i) * y(i,k)) =l= ck;
  DeliveryCapacity(l) ..     sum(i, q(i) * v(i,l)) =l= cl;


  PickupOnce(i)$(ord(i)<>1 and ord(i)<>8) ..
sum(k, y(i,k)) =e= 1;
  DeliveryOnce(i)$(ord(i)<>1 and ord(i)<>8) ..
sum(l, v(i,l)) =e= 1;

  NoYatStartDepot..      sum(k, y("0",k)) =e= 0;
  NoYatFinDepot..        sum(k, y("7",k)) =e= 0;

  NoVatStartDepot..      sum(l, v("0",l)) =e= 0;
  NoVatFinDepot..        sum(l, v("7",l)) =e= 0;




  TimeConstrPickup1(i,j,k)$(ord(j)<>1 and ord(j)<>8) ..
t_loaded(i,k) + A(i,j) + ELL(j)
- t_loaded(j,k) - M*(1 - x(i,j,k)) =l= 0;


  TimeConstrPickup2(i,k)$(ord(i)<>1)  ..
t_loaded(i,k) + A(i,"7") + TL(k) - T(k)
- M*(1 - x(i,"7",k)) =l= 0;


  TimeConstrDelivery1(j,l)$(ord(j)<>1 and ord(j)<>8) ..
R(l) + RL(l) + B("0",j) + ELL(j)
- r_unloaded(j,l) - M*(1 - z("0",j,l)) =l= 0;



  TimeConstrDelivery2(i,j,l)$(ord(i)<>1 and ord(i)<>8 and ord(j)<>1 and ord(j)<>8) ..
    r_unloaded(i,l) + B(i,j) + ELL(j)
- r_unloaded(j,l) - M*(1 - z(i,j,l)) =l= 0;



  TimeConstrDelivery3(i,l)$(ord(i)<>1) ..
r_unloaded(i,l) + B(i,"7")
- r_unloaded("7",l) - M*(1 - z(i,"7",l)) =l= 0;

  TimeConstrDelivery4(l) ..   r_unloaded("7",l) =l= Rmax;



  DeliveryPickup(i,k,l) ..    T(k) =l= R(l) + M * (-y(i,k)-v(i,l)+2);


  Tconstr(k)..      T(k) - M * (1-x("0","7",k)) =l= 0;
  Rconstr(l)..      R(l) - M * (1-z("0","7",l)) =l= 0;

  NoTime(i,k)..          t_loaded(i,k) =l= M * y(i,k);
  NoTimeDel(i,l)$(ord(i)<>8)..  r_unloaded(i,l) =l= M * v(i,l);
  runloadFinal(l)..     r_unloaded("7",l) =l= M * (1-z("0","7",l));

  AlfaSama(k,km) ..  alfa(k,km)$(ord(k)=ord(km)) =e= 0;
  BetaSama(l,lm) ..  beta(l,lm)$(ord(l)=ord(lm)) =e= 0;

  MultiUsePickup(k,km)..     alfa(k,km) =l= 1 + ((t_loaded("0",km)
```

```
- T(k))/Tmax) - x("0","7",k);
  MultiUseDelivery(l,lm)..   beta(l,lm) =l= 1 + ((R(lm)
- r_unloaded("7",l))/Rmax) - z("0","7",l);;

  AlfaConstr(k) ..       sum(km, alfa(k,km)) =l= 1;
  AlfaConstr2(km) ..         sum(k, alfa(k,km)) =l= 1;

  BetaConstr(l) ..       sum(lm, beta(l,lm)) =l= 1;
  BetaConstr2(lm) ..         sum(l, beta(l,lm)) =l= 1;

  NumberOfVehicles..      a_vehicles =e= sum((j,k)$(ord(j)<>8), x("0",j,k))
- sum((k,km), alfa(k,km));
  NumberOfVehiclesDel..   b_vehicles =e= sum((j,l)$(ord(j)<>8), z("0",j,l))
- sum((l,lm), beta(l,lm));

  TotSerTimeCalculated ..     TotSerTime =e= 2 * sum(i, ELL(i))
+ sum(k, TL(k)) + sum(l, RL(l));

  Model prodmix /all/ ;

prodmix.optfile = 1;
file cp /cplex.opt/;
put cp;
put "epint 1.0e-5" /;
putclose cp;
//----------------------------------------

//Solve prodmix using rmip minimizing Obj;

Solve prodmix using mip minimizing Obj;

DISPLAY x.L;
//DISPLAY x.M;
//DISPLAY x.LO;
//DISPLAY x.UP;

DISPLAY z.L;

DISPLAY y.L;
DISPLAY v.L;

DISPLAY t_loaded.L;
DISPLAY r_unloaded.L;

DISPLAY T.L;
DISPLAY R.L;

DISPLAY TL.L;
DISPLAY RL.L;

DISPLAY alfa.L;
DISPLAY beta.L;

DISPLAY a_vehicles.L;
DISPLAY b_vehicles.L;
```

# Appendix C

# Results from Parameter Tuning

The results from the parameter tuning are presented in the tables in this chapter. Each combination of parameters was run 5 times for each algorithm. The average value from these 5 runs along with the lowest and highest values are presented.

## C.1 Results for algorithm SA-A1

### C.1.1 Truck capacity 33

Table C.1: Results using data set `Day1_33`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 57396.2 | **57878.9** | 58669.3 | 57626 | **58150.7** | 58699.4 | 58480.2 | **58802.4** | 59226.1 |
| 100 | 59232.7 | **59631.7** | 59904.4 | 60015.8 | **60317.8** | 60798.5 | 61879.1 | **62304.7** | 62669.4 |
| 200 | 60817 | **61703.8** | 62633.3 | 61069.4 | **62039.7** | 63079.4 | 62678.9 | **63236.9** | 63585.7 |

Table C.2: Results using data set `Day1_160_33`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 34739.7 | **35108.5** | 35398.1 | 34894.5 | **35151.2** | 35636.5 | 35013 | **35451.2** | 35873.8 |
| 100 | 35081.2 | **35580.7** | 36216.7 | 35364.6 | **35673** | 36100.5 | 35688.4 | **35982.9** | 36231.5 |
| 200 | 35849.6 | **36058.3** | 36489.2 | 35768.2 | **36048.2** | 36527.2 | 35920.5 | **36318** | 36607.6 |

Table C.3: Results using data set `Day1_100_33`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 20804.8 | **21113.9** | 21431.1 | 21149.9 | **21348.4** | 21564.4 | 20840 | **21171** | 21449.9 |
| 100 | 20840.4 | **21160.7** | 21344.4 | 20826.2 | **21273.7** | 21636.8 | 21083.4 | **21265.1** | 21393.1 |
| 200 | 20945.6 | **21374.9** | 21720.1 | 20916.9 | **21380.9** | 21777.2 | 20955.1 | **21206.9** | 21486.9 |

### C.1.2 Truck capacity 50

Table C.4: Results using data set `Day1_50`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 44782.1 | **45433.1** | 46075 | 44951.8 | **45929.9** | 46500.4 | 45109.4 | **46219.1** | 47546.2 |
| 100 | 48112.8 | **48730.4** | 49785.8 | 48354.4 | **48746.1** | 49394.9 | 47912.8 | **48506.5** | 48995.5 |
| 200 | 48510.7 | **49380.1** | 50176.1 | 49495.6 | **50155.1** | 50671.4 | 49415.3 | **50034.9** | 50672.5 |

Table C.5: Results using data set `Day1_160_50`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 28323.9 | **28882.3** | 29363.5 | 28629.2 | **29223.4** | 29648.5 | 29261.1 | **29622.5** | 29910 |
| 100 | 30043.5 | **30393.3** | 30710.3 | 30374.3 | **30540.6** | 30685.7 | 30540.4 | **30778.5** | 31042.1 |
| 200 | 30597.1 | **30898.7** | 31093.6 | 30588.3 | **30765.7** | 30883.5 | 31072.9 | **31253** | 31355.8 |

Table C.6: Results using data set `Day1_100_50`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 17070.5 | **17610** | 17951.3 | 17231 | **17762.8** | 18310.2 | 17924.3 | **18177.8** | 18608.6 |
| 100 | 17810.6 | **18328.3** | 18780.1 | 18254.8 | **18645.9** | 19474.4 | 18492.6 | **18637.1** | 18951.4 |
| 200 | 18434 | **18735.4** | 18985.1 | 18547.6 | **18823.8** | 19066.1 | 17713.8 | **18189.8** | 18456.4 |

## C.2 Results for algorithm SA-A2

### C.2.1 Truck capacity 33

Table C.7: Results using data set Day1_33

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 55064.5 | **55495.4** | 56582.1 | 55011.8 | **55370.3** | 55740.9 | 56324 | **56553.5** | 56733.1 |
| 100 | 56909.3 | **57234.1** | 57705.8 | 57039.6 | **57571.8** | 58555 | 58664.2 | **59526.9** | 59958.3 |
| 200 | 58467.7 | **59091** | 59743.2 | 58332.9 | **59266.9** | 59831.7 | 60458.9 | **60991.8** | 61598.6 |

Table C.8: Results using data set Day1_160_33

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 33110.7 | **33355** | 33642.8 | 33260.8 | **33516.5** | 33907.1 | 33463.4 | **33631.1** | 33886.3 |
| 100 | 33528 | **33861** | 34484.9 | 33630.6 | **33976.1** | 34289.6 | 34185 | **34327.7** | 34393.6 |
| 200 | 34004.1 | **34205.6** | 34413.2 | 33963.7 | **34369.8** | 34755.7 | 34591.6 | **34861.4** | 35028.1 |

Table C.9: Results using data set Day1_100_33

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 19869.9 | **20054** | 20355.4 | 19786.5 | **19951.8** | 20167.9 | 19816.2 | **19919.3** | 19961.2 |
| 100 | 19763.3 | **20063.7** | 20274.3 | 19854.9 | **20040** | 20291.9 | 20037.3 | **20154.8** | 20353.1 |
| 200 | 20036.1 | **20298.9** | 20523.9 | 19952.1 | **20135.6** | 20422.3 | 19749.7 | **20128.9** | 20476.5 |

### C.2.2 Truck capacity 50

Table C.10: Results using data set Day1_50

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 44801.8 | **45466.6** | 46451.5 | 45876.3 | **46497.3** | 46843.7 | 45685.9 | **46388.1** | 47265.8 |
| 100 | 48188.3 | **48614.3** | 49079.8 | 48017.1 | **48791.3** | 49162.2 | 48647.9 | **49030.5** | 49309.8 |
| 200 | 49687.3 | **50187.8** | 50535.9 | 49945.7 | **50428.9** | 50957 | 48874.9 | **49882.1** | 50558.6 |

Table C.11: Results using data set `Day1_160_50`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 28277.3 | **28569.8** | 29107 | 28083.3 | **28835.6** | 29311.5 | 28557.8 | **29263.7** | 30282.3 |
| 100 | 29104 | **29712.1** | 30367.7 | 30152.7 | **30440.3** | 30705.8 | 30381 | **30880.7** | 31280.7 |
| 200 | 30552.7 | **30953.6** | 31320 | 30605.9 | **31184.1** | 31706.7 | 30809.4 | **31131.9** | 31621.4 |

Table C.12: Results using data set `Day1_100_50`

| | $\alpha = 0.75$ | | | $\alpha = 0.80$ | | | $\alpha = 0.85$ | | |
|---|---|---|---|---|---|---|---|---|---|
| T | Lowest | Average | Highest | Lowest | Average | Highest | Lowest | Average | Highest |
| 50 | 16655 | **16850.3** | 17010.7 | 16795.1 | **16927.7** | 17051.3 | 16987.1 | **17152.8** | 17344.7 |
| 100 | 16795.7 | **17052.6** | 17359.8 | 17398.8 | **17769** | 18058.1 | 18132.7 | **18500.7** | 18850.7 |
| 200 | 17193.3 | **17356.7** | 17520.1 | 17730.8 | **18000.8** | 18401 | 18673.7 | **18810.5** | 19083.2 |

# Appendix D

# Tables with temperature and time results

The tables in this chapter show the temperature at termination and the termination time. The algorithms checks every time it lowers the temperature the time elapsed. If it is more than 900 seconds the algorithm terminates otherwise it keeps on iterating.

## D.1 Temperature at termination of the algorithm and time

Table D.1: Temperature and time when algorithm **SA-A2** terminates

| Dataset | Ave Temp | Ave Time |
|---|---|---|
| Day2_33 | 6.00677 | 960 |
| Day3_33 | 5.00565 | 976 |
| Day4_33 | 6.67419 | 960 |
| Day5_33 | 5.67307 | 976 |
| Day2_160_33 | 2.00617 | 961 |
| Day3_160_33 | 3.00339 | 959 |
| Day4_160_33 | 2.14696 | 961 |
| Day5_160_33 | 2.00617 | 959 |
| Day2_100_33 | 0.229032 | 934 |
| Day3_100_33 | 0.126446 | 940 |
| Day4_100_33 | 0.261919 | 934 |
| Day5_100_33 | 0.140196 | 940 |

Table D.2: Temperature and time when algorithm **SA-A2** terminates

| Dataset | Ave Temp | Ave Time |
|---|---|---|
| Day2_50 | 28.125 | 1206 |
| Day3_50 | 28.125 | 1532 |
| Day4_50 | 30 | 1206 |
| Day5_50 | 34.375 | 1532 |
| Day2_160_50 | 29.6875 | 1021 |
| Day3_160_50 | 26.7188 | 1007 |
| Day4_160_50 | 21.0938 | 1021 |
| Day5_160_50 | 22.5 | 1007 |
| Day2_100_50 | 9.49218 | 992 |
| Day3_100_50 | 9.49218 | 985 |
| Day4_100_50 | 14.7217 | 992 |
| Day5_100_50 | 7.25722 | 985 |

## D.2 Results from comparing initial solutions

Table D.3: Results for data sets with truck capacity 33 using algorithm SA-A2 and a worse initial solution

| Dataset | Lowest | Average | Highest | Lower Bound | GAP |
|---|---|---|---|---|---|
| dataDay2 | 50527.5 | 51166.3 | 51983 | - | - |
| dataDay3 | 43727.1 | 44180.2 | 44462.7 | - | - |
| dataDay4 | 52295.9 | 53278.4 | 53828.9 | - | - |
| dataDay5 | 47753.6 | 48100.1 | 48453.1 | - | - |
| Average | | | | | |
| dataDay2_160 | 36416.6 | 36624.4 | 36895.1 | - | - |
| dataDay3_160 | 33819.9 | 34545.1 | 35112.7 | - | - |
| dataDay4_160 | 35229.8 | 35706.4 | 36004.7 | - | - |
| dataDay5_160 | 34531.7 | 35159 | 35422.9 | - | - |
| Average | | | | | |
| dataDay2_100 | 21746.8 | 22130.7 | 22557.8 | 7326.8 | 202.1 |
| dataDay3_100 | 21254.2 | 21431.7 | 21631.1 | 7296.4 | 193.7 |
| dataDay4_100 | 21033.1 | 21164.4 | 21360.5 | 7245.2 | 192.1 |
| dataDay5_100 | 21765.7 | 21989.6 | 22254.1 | 7483.6 | 193.8 |
| Average | | | | | 195.4 |

Table D.4: Results for data sets with truck capacity 33 using algorithm SA-A2 and a worse initial solution

| Dataset | Lowest | Average | Highest | Lower Bound | GAP |
|---|---|---|---|---|---|
| dataDay2 | 51598.8 | 52180.5 | 52683.9 | - | - |
| dataDay3 | 47179.7 | 47656.7 | 48531 | - | - |
| dataDay4 | 58472.4 | 59021.4 | 59459.2 | - | - |
| dataDay5 | 50141 | 50743.6 | 51104.8 | - | - |
| Average | | | | | |
| dataDay2_160 | 38147.8 | 38676.5 | 39355.9 | - | - |
| dataDay3_160 | 37072.6 | 37592.2 | 38241 | - | - |
| dataDay4_160 | 38517.5 | 39663.3 | 40615.4 | - | - |
| dataDay5_160 | 37144.3 | 37276.1 | 37514.4 | - | - |
| Average | | | | | |
| dataDay2_100 | 23353.7 | 23878.7 | 24395.4 | 7326.8 | 225.9 |
| dataDay3_100 | 22649.9 | 23065.4 | 23575.5 | 7296.4 | 216.1 |
| dataDay4_100 | 22425 | 23071.9 | 23807.6 | 7245.2 | 218.4 |
| dataDay5_100 | 23306.9 | 23487.3 | 23641.3 | 7483.6 | 213.9 |
| Average | | | | | 218.6 |

# Appendix E

# Figures showing the routes

The figures in the chapter display the routes geographically. In this example there were 32 pickup routes and 32 delivery routes. For the pickup and delivery routes there are 4 figures, each showing eight routes. To make the routes clearer, lines from first and last order to the the cross dock are not shown.

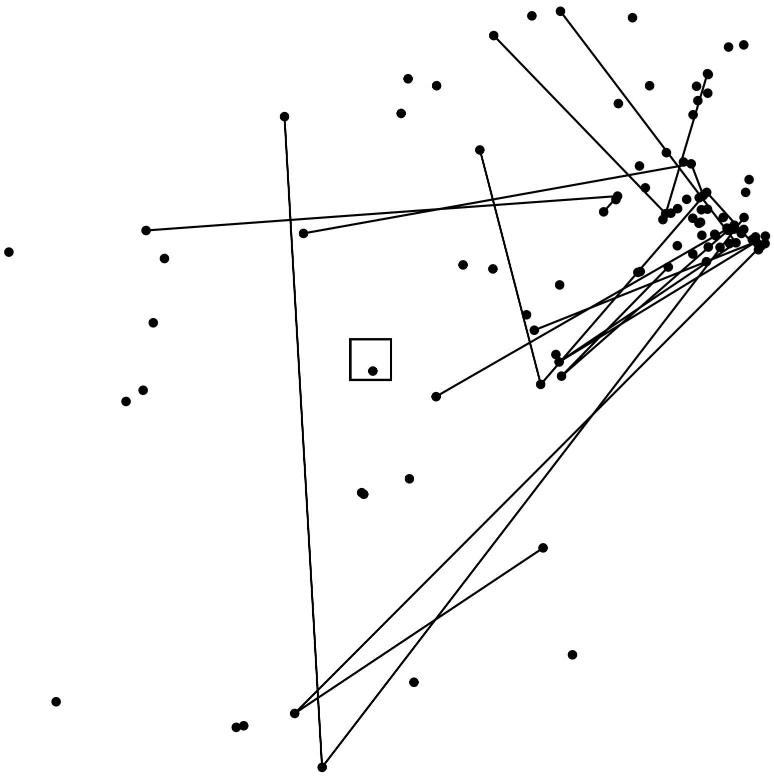

Figure E.1: Geographical display of pickup routes, routes 1-8

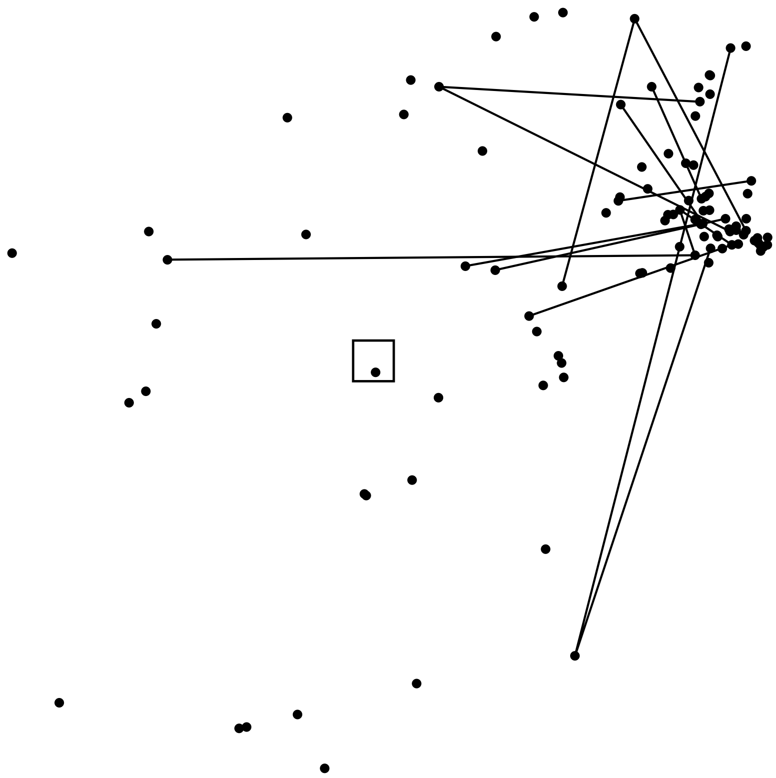

Figure E.2: Geographical display of pickup routes, routes 9-16

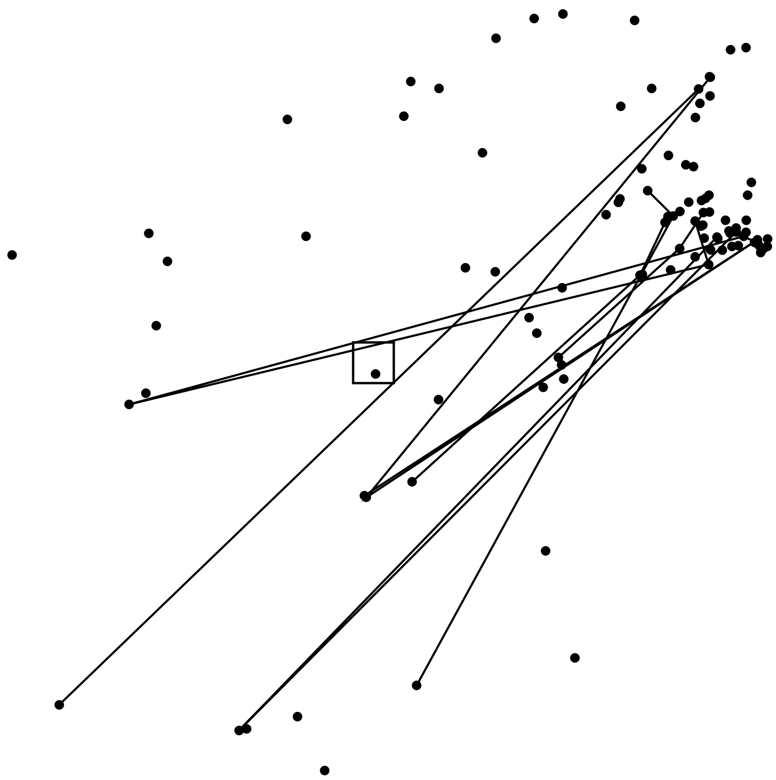

Figure E.3: Geographical display of pickup routes, routes 17-24

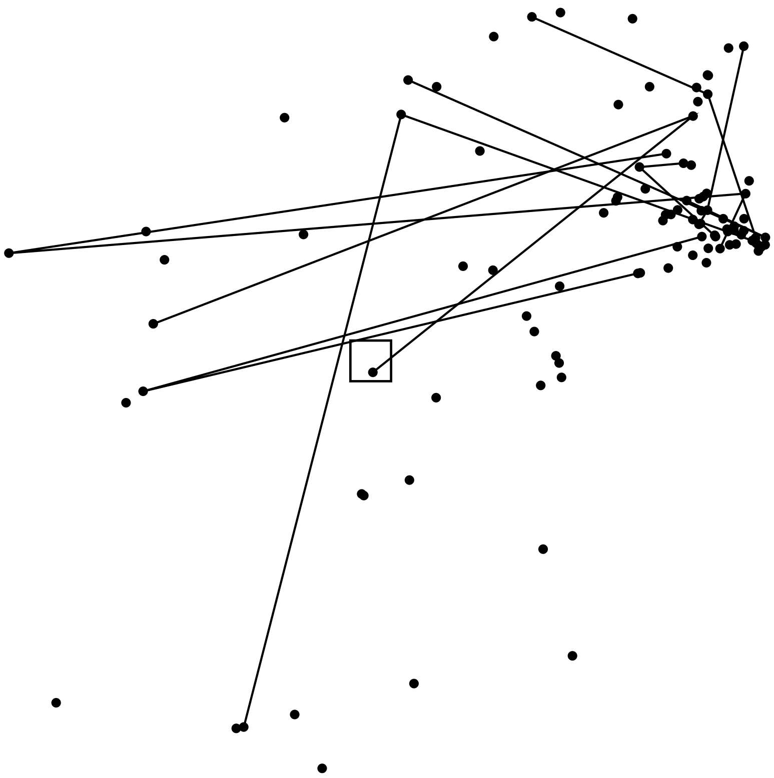

Figure E.4: Geographical display of pickup routes, routes 25-32

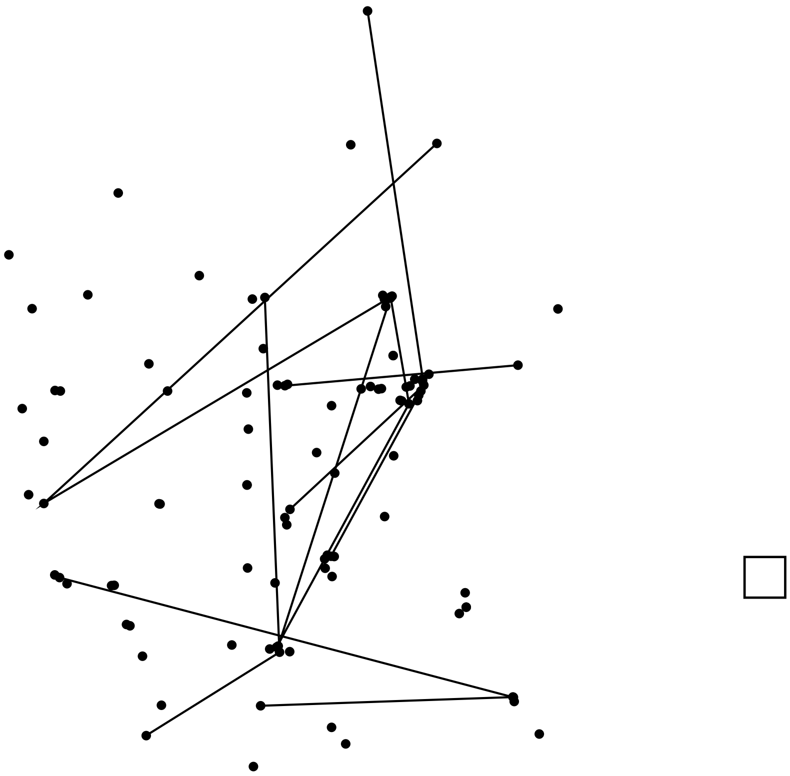

Figure E.5: Geographical display of delivery routes, routes 1-8

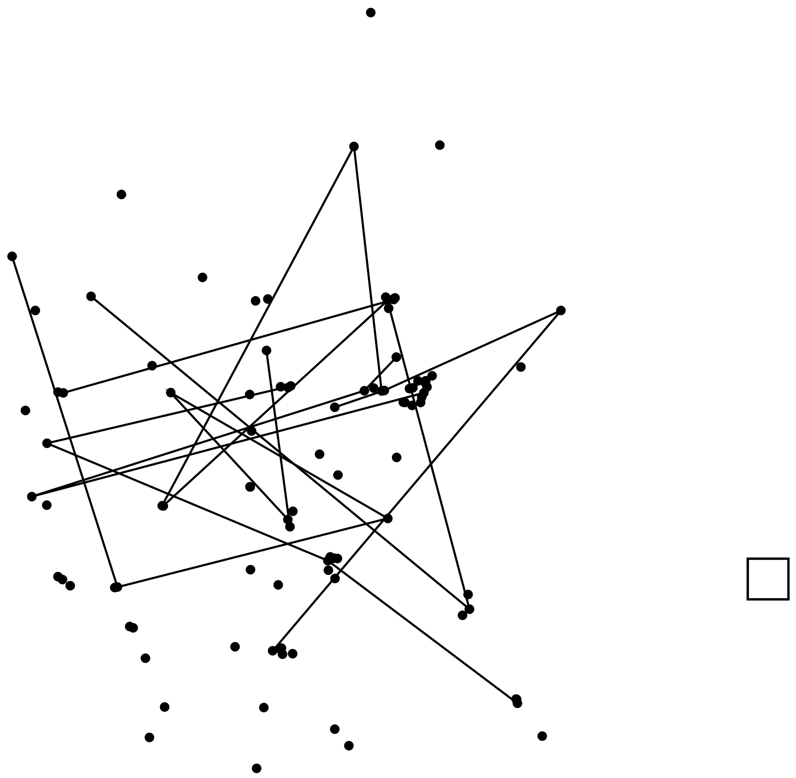

Figure E.6: Geographical display of delivery routes, routes 9-16

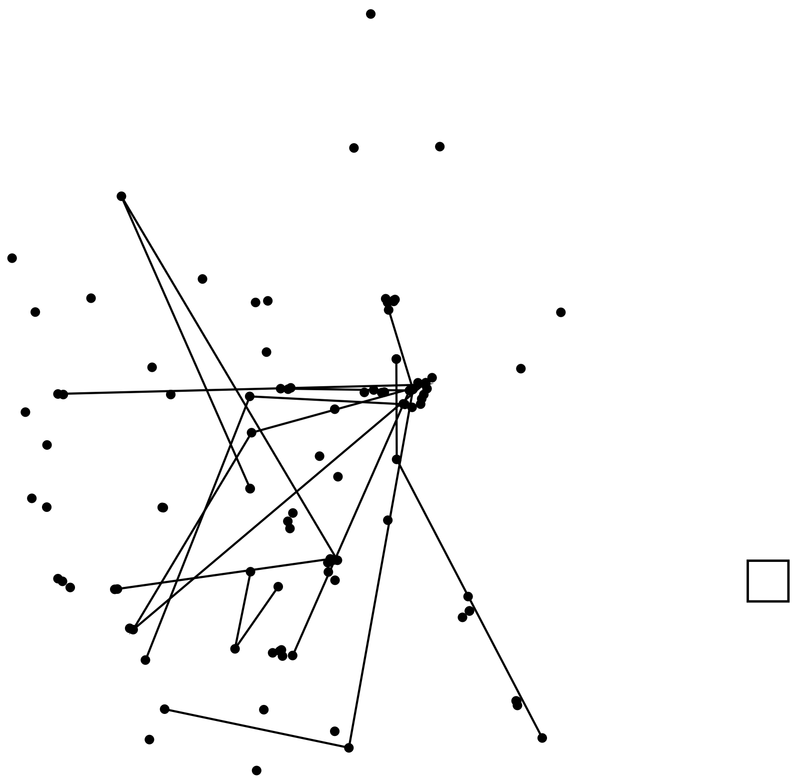

Figure E.7: Geographical display of delivery routes, routes 17-24

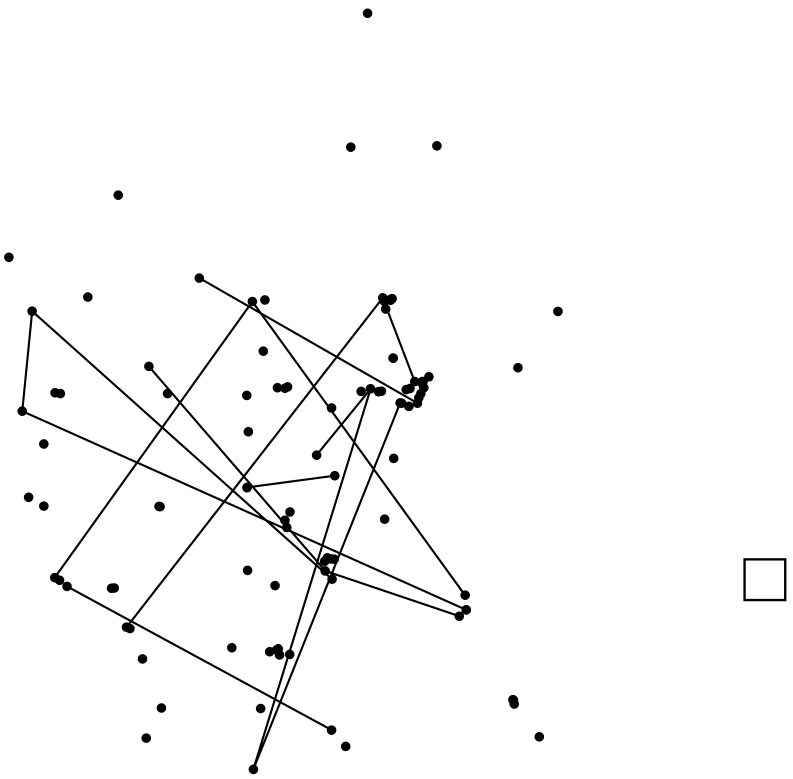

Figure E.8: Geographical display of delivery routes, routes 25-32