UNIVERSITY OF COPENHAGEN
DEPARTMENT OF COMPUTER SCIENCE

MASTER THESIS

Solving VRP using Voronoi Diagrams and Adaptive Large Neighborhood Search

Author:
NIELS PETER MEYN MILTHERS (MILTHERS@DIKU.DK)

Supervisors:
DAVID PISINGER (PISINGER@DIKU.DK)
BJØRN PEDERSEN (BJORN@DIKU.DK)

January 29, 2009

Abstract

In this thesis I create an algorithm to solve the Vehicle Routing Problem with Time Windows. This algorithm will use the clustered structure of many real world datasets caused by large cities. This structure is used to divide the problem into subproblems which can be solved before they are combined and the solution is improved. The algorithm should be able to solve the largest benchmark problems quickly, i.e. within 10 minutes.

The main contribution of this thesis is the initial solution created to start the Adaptive Large Neighborhood Search, together with the use of regions in the ShawRemoval algorithm.

The algorithm proposed in this thesis shows promising results, but improvements are possible.

I dette speciale konstruerer jeg en algoritme til at løse Ruteplanlægnings Problemet med Tidsvinduer. Algoritmen bruger den klyngestruktur der ofte kan findes i realistiske problemer på grund af store byer. Denne struktur bruges til at splitte problemet op i delproblemer der kan løses inden de kobles sammen og løsningen søges forbedret. Algoritmen skulle være i stand til at løse de største benchmark problemer hurtigt, dvs. inden for 10 minutter.

Det vsentligste bidrag fra dette speciale er den indledende løsning der bruges i starten af den Adaptive Store OmegnsSøgning, samt brugen af regioner i Shaws kundefjerningsalgoritme.

Algoritmen der foreslås i dette speciale viser lovende resultater, men der er muligheder for forbedringer

Contents

Chapter 1

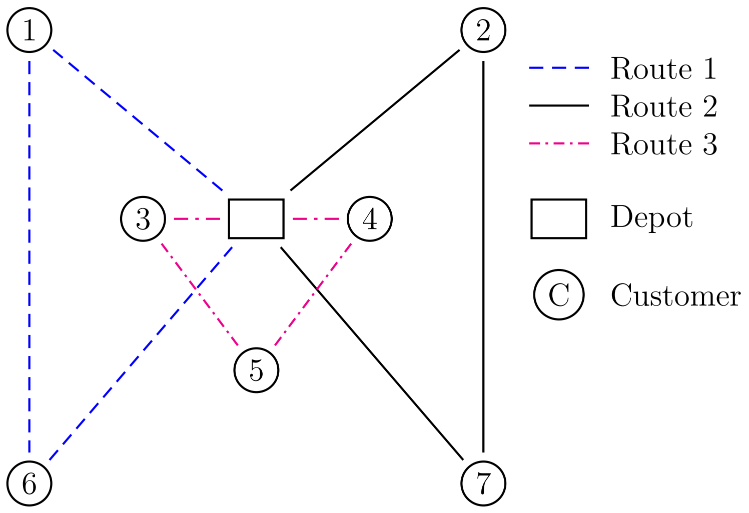
Introduction

In this Master thesis I solve the Vehicle Routing Problem with Capacity constraints (VRP). The problem can be described as: A company has a set of customers all having a demand of goods. The customers must be served by a fleet of vehicles, all leaving and returning to the same depot. Each customer must be visited exactly once, and each vehicle can only carry a certain amount of goods that must not be exceeded. The transportation between each pair of nodes, the customers and the depot, has a certain cost associated with it. The cost can be both time consumption and traveling distance, and the objective is to minimize the total cost of the routes.

The VRP problem has been solved in many ways since Dantzig and Ramser [1] described it in 1959. While solving the VRP can be useful for some problems such as picking up garbage, it is often necessary to give the customers some other information. In order to give this extra information, some extra constraints have been added to the problem to make it more realistic, and more useful in real world applications.

When the customers require the vehicle to arrive within a certain time window, these time windows can be added to the model. If a vehicle arrives at a customer before the start of the time window, the vehicle must wait until the time window starts before it can unload the goods. The vehicle may not arrive later than the end of the time window. The Time Window constrained Vehicle Routing Problem (VRPTW) is the main focus of this thesis. Pullen and Webb [2] were among the first to define the VRPTW. When dealing with time windows, the time usage for traversing an edge must be available, often the time usage is the same as the traveling distance, to simplify the data. A solution to a simple VRP instance is shown in figure 1.1.

Figure 1.1 A Solution to the VRP



The remainder of this chapter is structured as follows. In section 1.1 I formulate the problem at hand more formally, in section 1.2 I give a short survey of previous approaches to the problem before I describe the scope of the thesis in section 1.3. In section 1.4 I describe the Voronoi Diagrams used in the thesis briefly before I give an outline of the rest of the thesis in section 1.5.

1.1 VRP formulation

The problem can be described more formally as: Let C be the set of customers and let the set of nodes be $V = C \cup \{o, o'\}$, where $\{o\}$ is the depot at the beginning of the routes and $\{o'\}$ is the depot at the end of the routes. $E = \{(i, j) : i, j \in V, i \neq j\}$ denotes the set of edges between the nodes. Let K be the set of vehicles, with $|K|$ unbounded. Each vehicle has the capacity D_k, and each customer $i \in C$ has a demand d_i, with the demands for the two depots $d_o = d_{o'} = 0$. For each node there is a beginning (a_i) and an end (b_i) to the time window where the vehicles are allowed to visit the node. Let s_i be the service time for $i \in V$ and t_{ik} be the time vehicle $k \in K$ is at node $i \in V$, if k visits i. Let c_{ij} be the cost of traversing the edge $(i, j) \in E$ and h_{ij} be the time used to traverse the edge. Let x_{ijk} be a variable denoting whether vehicle $k \in K$ traverses the edge $(i, j) \in E$. Finally let $\tau_{ij} = h_{ij} + s_i$ be the travel time on edge $(i, j) \in E$ plus the service time for customer i.

When this is modeled mathematically in the three index flow model it looks like this, as described by Toth and Vigo [3].

$$\begin{aligned}
&\min \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ijk} && && (1.1)\\
&\text{S.T} \sum_{k \in K} \sum_{(i,j) \in \delta^+(i)} x_{ijk} &=& \; 1, \forall i \in C && (1.2)\\
&\sum_{(i,j) \in \delta^+(o)} x_{ijk} = \sum_{(i,j) \in \delta^-(o')} x_{ijk} &=& \; 1, \forall k \in K && (1.3)\\
&\sum_{(i,j) \in \delta^+(i)} x_{ijk} - \sum_{(i,j) \in \delta^-(i)} x_{ijk} &=& \; 0, \forall k \in K, \forall i \in C && (1.4)\\
&\sum_{(i,j) \in E} x_{ijk} \cdot d_i &\leq& \; D_k, \forall k \in K && (1.5)\\
&a_i \leq t_{ik} &\leq& \; b_i, \forall i \in V, \forall k \in K && (1.6)\\
&x_{ijk}(t_{ik} + \tau_{ij}) &\leq& \; t_{jk}, \forall (i, j) \in E, \forall k \in K && (1.7)\\
&x_{ijk} &\in& \; \{0, 1\}, \forall (i, j) \in E, \forall k \in K && (1.8)
\end{aligned}$$

$\delta^+(i)$ is the set of edges leaving the node i, and $\delta^-(i)$ is the set of edges entering the node i.

The objective function (1.1) of the problem is in this case to minimize the total length of the routes. The constraints (1.2) ensures that each customer is visited by exactly one vehicle. Constraints (1.3) ensures that the number of vehicles leaving and entering the depot is the same. To preserve the flow of the network the constraints in (1.4) says that each vehicle that visits a customer should leave that customer again. Each vehicle has a maximum load it can carry, constraints (1.5) takes care of this. Constraints (1.6) and (1.7) ensure that the time windows of the customers are satisfied. Finally constraints (1.8) says that all the x variables must be binary.

Subtours are eliminated by the time window constraints (1.7), otherwise both the travel times τ_{ij} and τ_{ji} would be zero and this would only happen if the two customers were in the same place and no time would be used to unload goods.

Solomon [4] has introduced a series of test instances which are used to compare the results in this thesis. For the past years all the instances by Solomon except 5 are solved, therefore new versions have been made, with up to 1000 customers. These instances are due to Homberger [5].

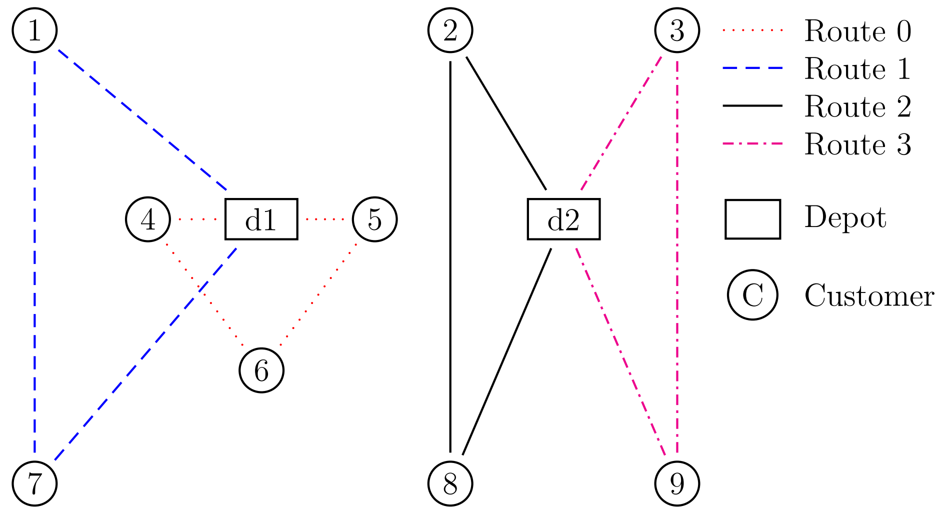
In order to cope with larger problem instances, I will divide the problem into smaller subproblems. These subproblems will be solved, and then I will combine the solutions in order to get a solution to the original problem.

The subdivisions of the problem can be done in several ways. When a solution is made, it will most likely be better to have customers located near each other in the same route. This seems to advocate for a division where the location of the customers is considered.

Other versions of VRP

When the VRP problem has more than one depot it is possible to divide the problem into several smaller problems surrounding each depot, or it is possible to solve the problem as one problem. This problem is known as the Multiple Depot Vehicle Routing Problem, MDVRP. A solution to a MDVRP instance is shown in figure 1.2.

Figure 1.2 An instance of the MDVRP



The MDVRP is mentioned because it is the inspiration to the solution method described in this thesis.

1.2 Previous approaches

The VRPTW algorithms can be divided into two main categories, exact and heuristic solutions. The exact algorithms are mainly different versions of Branch and Bound algorithms, where the integrality of the variables are relaxed and solved as linear programming problems. When a solution to the LP-formulation of the problem is found the solution space is divided into two separate parts described by one of the variables being integer. One version is the Branch And Cut algorithm (BAC) which uses cuts to forbid combinations of legal routes to be chosen together. The BAC algorithm was described by Bard et al. [6].
Another version is the Branch and Cut and Price algorithm (BCP) for the VRPTW. The BCP algorithm only looks at a part of the legal routes at the beginning. It creates new routes which are priced into the solution and cuts illegal combinations out. This approach was proposed by Desrosiers et al. [7] and further developed by Desrochers et al. [8], Kohl [9], Larsen [10], Cook and Rich [11] and Irnich and Villeneuve [12]. Jepsen et al. [13] introduced new cuts and solved many of the previously unsolved benchmark instances.

The heuristics can be divided into several categories, among these are construction heuristics, which construct the routes from scratch with no optimization of the produced routes at the end, a sequential route creation version of this was proposed by Solomon [14]. A version where the routes are created in parallel is presented by Potvin and Rousseau [15]. Improvement heuristics uses the results from the construction heuristics and make substitutions where a pair of customers from different routes are exchanged. Usually when both types of heuristics are implemented the heuristic is defined as a two phase heuristic.

The creation heuristics have at least two variants. One variant is the insertion heuristics where the routes are created by inserting each customer in the place it increases the cost the least. An other variant is savings heuristics where each customer is inserted to its own route in the beginning, and the heuristic combines the two routes that gives the greatest decrease in the total cost until no more routes can be combined. I will describe some of these heuristics in greater detail in section 2.1.

In later years the heuristical focus has been on metaheuristics, where a heuristic is used to choose which heuristic should solve the problem, and there has even been Hyper Heuristics as described by Burke et al. [16]. One of the later metaheuristics is the Adaptive Large Neighborhood Search (ALNS) algorithm proposed by Røpke and Pisinger [17]. Røpke and Pisinger [17] use the ALNS to solve the Pickup and Delivery Problem with Time Windows (PDPTW) which is related to the VRPTW, but also considers the pickup of the goods. The ALNS is a Large Neighborhood Search (LNS) algorithm, where every iteration uses the results of the previous iterations in order to make better choices for the selection of Neighborhoods to search. The LNS approach was proposed by Shaw [18] and further developed in Shaw [19].

1.3 Scope of the thesis

In this thesis I propose a solution method for the VRPTW problem, in which I take advantage of the clustering of customers often found in regions with many islands, or mountains. The general idea of the solution method is to divide the problem into subproblems in order to solve these problems locally, and then combine these local solutions into a grand solution. The theory behind this solution method is that routes do not start in one corner of the plane and travel through the entire plane, but will stay in a minor part of the plane.

If some knowledge of the underlying structure of the geography or infrastructure is known, the problem can be divided into strongly connected parts and then solved in these parts. A good set of starting points for routes could be infrastructural hubs, such as a freeway crossing or similar. The idea is to find customers who are close to such a hub, or to find hubs that are close to several customers.

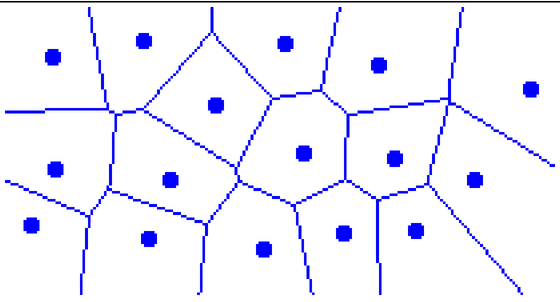
When the problem is solved in the regions, there will most likely be routes in which vehicles are not entirely filled, these vehicles could be combined in order to minimize the total number of vehicles used. In order to combine these vehicles in a clever way, the division could be tested to see if some pairs of regions would make bad combinations. Such bad combinations could be two regions only combined by a long bridge, or divided by a huge mountain. On the other hand, regions with many small routes combining them could be good candidates to combine routes with half filled vehicles from.

When no information of the geography or infrastructure for the problem is known, some other means to find strongly connected components has to be used. For this I propose to use Voronoi Diagrams.

1.4 Voronoi Diagrams

A Voronoi Diagram is a division of the plane into regions where each point in a region is closer to the defining point of the region, than the defining point of any other region, see figure 1.3.
de Berg et al. [20] describes the problem as the post office problem, i.e. which customers are closest to a given post office and where are those who have the longest distance to any post office. The Voronoi Diagram can be used to decide where a new post office should be placed, or it can be used to determine which customers should be served by which post office. The Voronoi Diagrams also give information about which of the post offices are neighbors. As the Voronoi Diagram is a planar graph, it has some very useful characteristics, which I will describe and use in section 4.3.

Figure 1.3 A Voronoi Diagram



Voronoi Diagrams have been used to identify neighbors in a insertion heuristic, in order to improve the placement of the customers, this algorithm is created by Kwon et al. [21]. Beasley and Christofides [22] use Voronoi Diagrams, to make an adjacency list of the customers, in order to create a feasibility graph.

Voronoi Diagrams have to my knowledge not been used to solve the VRPTW by dividing the problem into subproblems.

1.5 Structure of the thesis

In chapter 2 I will describe the solution methods used to solve the problem, in chapter 3 I will describe the adaptive structure used to explore the solution area. The initial solution of the problem, used in the metaheuristic of chapter 3, will be described in chapter 4. In chapter 5 the tuning of the algorithm is described together with the computational results. Finally in chapter 6 there will be some discussion of what could be done to improve the algorithm, and some concluding remarks.

I would like to thank the company Transvision for the real world data used in this thesis.

Chapter 2

Solution methods

In this chapter I will describe some of the heuristic solution methods previously used and those I will use to solve the VRPTW.

As mentioned in section 1.2, simple route creation heuristics fall into two categories: one is insertion heuristics, where each customer is inserted into a route, until all customers are placed. The other category is combination heuristics, where each customer is initially its own route, and then pairs of routes are combined until no more routes can be combined. In section 2.1 I will describe some of the more common insertion heuristics, and in section 2.2 I will describe how the solutions produced by the heuristics can be improved. Section 2.3 gives a brief introduction to metaheuristics and in section 2.4 I will describe a specific metaheuristic, which will be extended in chapter 3.

I will not describe any combination heuristics, they are mainly used to solve VRP and are not well suited for the VRPTW metaheuristic I propose to use later in the thesis. The reason the combination heuristics are not well suited for the VRPTW is when time windows are to be considered, the best combination of two routes may make it impossible to combine that route with other routes later which they would otherwise be able to.

2.1 Insertion heuristics

Insertion heuristics can be divided into two classes, namely sequential and parallel. These terms should not be confused with the similar terms for processor usage. A sequential algorithm builds one route at the time, whereas the parallel algorithms build several routes at the same time. A description of a sequential algorithm can be seen later in this section.

Calculating insertion values
The insertion algorithms described in this section all use a routine to calculate the value for inserting a customer c_1 to a given route r_1. The best place in the route to insert the customer is found and the increase in cost is used as the measure for how good the insertion is.
Instead of calculating the best placement in the route, the cost increase for inserting the customer at the end of the route could be found. I believe that the calculation of the best place to insert the customer in the route is not too time consuming, and will lead to better routes. If the customer can not be inserted into the given route, I set the cost of inserting it to ∞.

Greedy insertion
Most insertion heuristics have some form of greediness built into them, the simplest greedy heuristic is shown in algorithm 2.1. The basic greedy algorithm will often end up with the customers which are hard to place in one route, thus placing them when their best placement is not available anymore. The other algorithms try to address this problem by looking a bit into the future and insert these customers into the routes earlier in the process.

The basic greedy insertion heuristic shown in algorithm 2.1 starts with an empty set of routes in line 1. The while loop in lines 3 to 10 finds the best customer to place in a route in line 4. If the customer should be inserted to a new route this route is created in lines 5 and 6.

The greedy insertion heuristic implemented for this thesis only keeps track of the best placement for a customer c_i in the current solution. This implies that a new relation value for a customer c_i has to be calculated every time another customer c_j is inserted into the route r_i that the customer c_i has its best insertion value in. Thus calculating the insertion costs several times,

Algorithm 2.1 Basic greedy insertion heuristic

```
GREEDY(customer List cList)
 1  Routes ← $\emptyset$
 2  generate insertion values for the customers in the empty route
 3  **while** cList $\neq \emptyset$
 4          **do** find customer $c1$ with lowest insertion value
 5              **if** c1 should be inserted into a new route
 6                  **then** create a new route r1 and insert $c1$
 7                           Routes ←Routes $\cup$\{r1\}
 8                  **else**  insert $c1$ to the wished route
 9              cList ← cList\\{$c1$\}
10              generate new insertion values for cList
11  **return** Routes
```

but without using a lot of memory, which could be useful when dealing with many customers and many routes.

In addition to the basic greedy insertion algorithm, I have created a heuristic which remembers the insertion values for each route and customer, instead of recalculating them every time the route is changed.

I will test these two algorithms, to see if the speedup from the memory spending algorithm is worth it, or if the memory usage will become a problem.

Regret insertion heuristics
In order to compensate for the bad placement of the last customers, a regret algorithm can be used. This algorithm looks at the best place to put the customer, and the next best placement of the customer in another route. The customer with the biggest difference between the two best placements is inserted. This way the customers that are difficult to place, when only looking at the best placement, should be placed earlier in the process. If tied, the customer with the lowest increase in cost will be inserted. The regret algorithm has been used previously to solve the VRPTW by Potvin and Rousseau [15].

The regret heuristic maximizes the difference in insertion values $f_0(c) - f_1(c)$, where $f_i(c)$ is the i'th best place to put customer c.

$$\max_{c \in C} \quad f_0(c) - f_1(c) \tag{2.1}$$

Algorithm 2.2 Regret-k Algorithm

```
REGRET(customer List cList, int k)
1  Routes ← $\emptyset$
2  while cList $\neq \emptyset$
3          do find customer with highest regret-$k$ value and insert it to route r1
4             calculate new insertion values for route r1
5             if r1 is a new route
6                then add it to Routes
7  return Routes
```

The regret heuristic can be extended to look at the k'th best place to put the customer, instead of just the second best place, i.e. to look at the difference $f_0(c_i) - f_k(c_i)$. Ties are broken by regarding the regret value for waiting $k-1$ iterations before inserting the customers. The memory using greedy algorithm can be described as a regret-1 algorithm, because it only look for the best place to put the customer. The regret algorithm could also be named regret-2.

The regret algorithms could look at several insertions in the same route, but as the routes would change after each insertion, the calculations would have to be redone for many placements of the customer, and not just one after each insertion.

Sequential greedy insertion heuristic

The Sequential greedy heuristic in algorithm 2.3 creates one route at the time, by selecting a seed customer and starting the route by inserting this customer to the empty route, line 4. The while loop, lines 3-9, continues until all customers have been placed in a route. The inner loop, lines 6-8, calculates the cost for inserting the remaining customers into the current route, and chooses the best customer to insert, if no more customers can be inserted into the route, the route is saved in line 9. If there are still customers left, not yet included in a route, a new seed customer is found, and the inner loop is run again.

The sequential algorithm is inspired by the IMPACT algorithm developed by Ioannou et al. [23], but with a much simpler calculation of the best placement of the customers. Ioannou et al. [23] takes into consideration the effect it will have on later insertions to place a customer at the given place, and as such tries to cope with the problem of inserting the worst customers last.

Algorithm 2.3 Sequential greedy heuristic

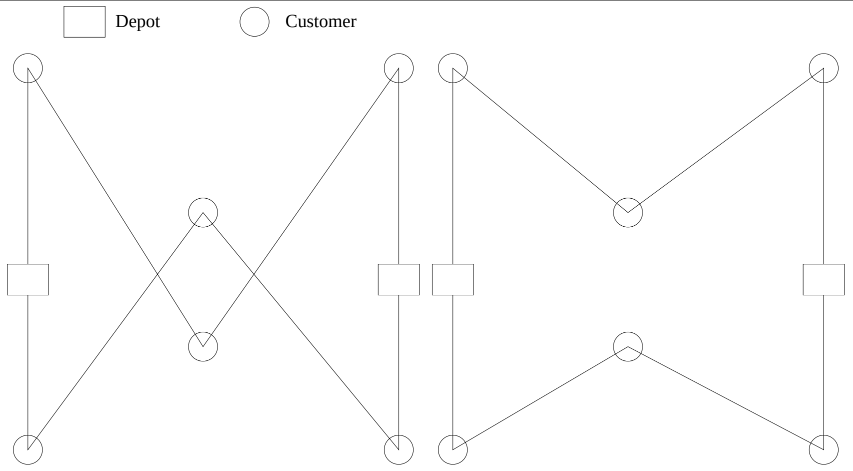
```
SEQGREEDY(customerList cList)
1  Routes ← $\emptyset$
2  generate insertion values for the customers in the empty route
3  while cList$\neq \emptyset$
4        do choose seed customer
5           insert seed customer to a new route $r$
6                 do calculate insertion values for the remaining customers
7                    find best customer to insert and insert into the route
8              until no more customers can be inserted to the current route
9           Routes = Routes $\cup\{r\}$
10 return Routes
```

2.2 Optimization heuristics

Often when the routes have been created, it could be useful, to optimize the routes. Heuristics for this optimization is often referred to as *two phase* heuristics, as they create the routes in the first phase by one of the creation heuristics mentioned in section 2.1. This initial solution is then altered in the second phase, until no more improvements can be obtained. These improvements are often done by swapping a pair of customers, see figure 2.1, or by moving one customer between two routes.

If the new solution created by moving customers have a better value than the previous one, the new solution is kept, otherwise the old solution is used again. The optimization continues untill no improvements can be made for the current solution. The problem with this strategy is that the initial

Figure 2.1 Illustration of the exchange move



solution can be near a local optimum, and thus work towards this optimum, which can be very bad in relation to the global optimum. In order to move away from these local optima, metaheuristics have been created.

2.3 Metaheuristics

In the heuristics described in section 2.2, the altered solution is only accepted if it has improved the solution value. In order to examine a larger section of the possible solutions, solutions can be accepted with some probability according to the solution value, or if the solution has not been visited in a period of time.

Neighborhoods

One of the keywords in metaheuristics is the neighborhood $N(s)$ of the solution s. The neighborhood is those solutions which can be made from the current solution by changing it. The change could be small, the movement of one or two customers, or it could be large, moving half the customers or more. A neighborhood for a solution, is the other solutions which can be made with the route changing algorithms. The altered solution is named s'

Simulated Annealing
Simulated Annealing (SA) simulates the heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The algorithm is greedy as it accepts a new solution if it is better than the previous solution. If the new solution is worse than the previous solution, it is accepted with a probability that decreases over time, simulating the cooling of material. The SA approach was described independently by Kirkpatrick et al. [24] and Černý [25].
An acceptance criteria often used is to set the probability p of accepting the solution to be the exponential function lifted to the difference in cost divided by the temperature T, when minimizing the cost, the exponent is negated.

$$p = e^{-\frac{f(s')-f(s)}{T}} \tag{2.2}$$

Figure 2.2 Graph of the exponential function

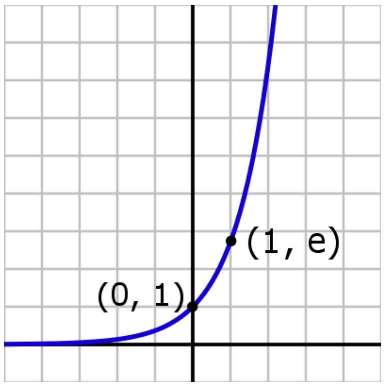


Figure 2.2 shows the exponential function. It shows that when the exponent $-\frac{f(s')-f(s)}{T}$ is below 0, the possibility becomes a value between 0 and 1, and if the exponent is larger than 0, the possibility is larger than 1 and will always be accepted. In equation 2.2 this means, that if the solution value to the new solution $f(s')$ is lower than the solution value of the old solution $f(s)$, the solution will be accepted. Otherwise the new solution will be accepted at a probability decreasing while the temperature T decreases. In each iteration the temperature is lowered by a factor $0 < c < 1$, the closer c is to 1, the slower the cooling will be.

TABUSearch
TABUSearch (TS) searches the neighborhood of the current solution, and

takes the best solution in the neighborhood, not considering the current. To avoid cycles, the TS keeps a record of solutions already visited, and will not visit solutions it has visited in the past n iterations. For the VRP, the tabu could also be a set of customer pairs, or arcs, which are not allowed to use in a solution, untill sufficient time has passed. The TS was proposed by Glover [26] and continued in Glover [27].

2.4 Large Neighborhood Search

The Large Neighborhood Search Algorithm (LNS) tries to move further around in the solution space in each iteration than the local search heuristics. The LNS moves a large number of customers in each iteration, by selecting and removing customers that are alike, and inserting them again, hoping that the customers will be easier to move around when the customers potentially could be inserted in all the places where a customer was removed.

In order to make new solutions that could be better than the ones created from the initial heuristics, one or more customers must move from one route to another. If the geography of Denmark is considered, the many islands of Denmark constitutes a problem in this strategy. Suppose a small island have two or more customers on it, and one of the customers is removed, then the resulting routes will all have the cost of traveling to and from the island, hence it would almost always be a good idea to keep these customers on the same route, this leads to the use of Large Neighborhood Search.

The LNS algorithm in algorithm 2.4 takes as input a set of possible solutions which could be only one solution and a natural number q defining how many customers should be moved in each iteration. The chosen starting solution is set to be the overall best solution in line 1. The loop from line 4 to 10 removes q customers in line 4 using any removal algorithm and inserts them again using an insertion algorithm in line 5. If the created solution is better than the current best solution, the best solution is updated in line 7. In line 8 it is decided if the current solution should be used in the next iteration, the decision is made based on how good the solution is and a randomization factor. The loop ends when a stop criterion is met, it could be that the solution has not improved in a number of iterations or that the maximum number of iterations is reached.

Algorithm 2.4 Large Neighborhood Search

```
LNS($s \in \{solutions\}, q \in \mathbb{N}$)
1   solution $s_{best} = s$
2   repeat
3           $s' = s$;
4           remove $q$ customers from $s'$
5           reinsert removed customers into $s'$
6           if ((f($s'$) $<$ f($s_{best}$))
7              then $s_{best} = s'$
8           if accept($s', s$)
9              then $s = s'$
10      until stop-criterion met
11  return $s_{best}$
```

The Large Neighborhood proposed by Shaw [18] removes related customers, in order to make the solution move around the solutions space. A randomness factor is added, this means that if the program is run several times, the solution will not necessarily be the same. This ensures that the solution space is more thourougly searched. The neighborhood proposed by Shaw will be described in detail in section 3.1.

In LNS each iteration consists of a series of removals from the routes and a series of insertions into the routes. If the new solution is better than the previously best, the new solution is set as the new best solution.

The solution is kept and used to work on in the next iteration, using any of the previously mentioned metaheuristics.

The LNS terminates when a predefined number of iterations has been run.

Chapter 3

Adaptive Large Neighborhood Search

In this chapter I will present the Adaptive Large Neighborhood Search, which is an extension of the LNS algorithm presented in section 2.4. Instead of using only one Large Neighborhood, ALNS uses several neighborhoods. A neighborhood of a solution will in this chapter consist of one removal phase and one insertion phase. The Removal phase chooses a removal algorithm to choose which customers should be removed, and the insertion phase chooses which insertion algorithm should be used to insert the removed customers. The removal algorithms will be described in section 3.1, and the insertion algorithms are basically the ones described in section 2.1, but they are altered slightly, these alterations are described in section 3.2.

In order to better choose which algorithms to be used in the removal and insertion phases, a system of adaptivity is added to the algorithm, this adaptivity procedure is described in section 3.3. If there was no adaptivity procedure the algorithm could be called Multiple Large Neighborhood Search.

Like the LNS algorithm, the ALNS algorithm starts with a solution to the problem. This solution could be created in many ways, for instance choosing one of the insertion algorithms described in section 2.1. But I have created an initial solution method which tries to use the clusterings of customers in many real life problems as an advantage. I will describe the creation of the initial solution in chapter 4. For the remainder of this chapter I will assume that an initial solution is present, and therefore will only go into detail with the alterations of the solution, and how I find the final solution.

3.1 Removal heuristics

When a solution is presented, and it shall be altered, first the customers to be moved around in a given iteration must be chosen. The algorithms presented in this section chooses the customers in different ways, this should make the ALNS more likely to adapt to different problem structures.

RandomRemoval
The RandomRemoval algorithm removes $q \leq |C|$ randomly chosen customers from the routes, with no regard to the cost benefit of removing the customers. This algorithm should in itself not present a good solution, but by using this removal algorithm, the algorithm should be able to search the solution space more extensively than if only the following removal algorithms were used.

ShawRemoval
ShawRemoval which was presented by Shaw [18] is shown in algorithm 3.1. ShawRemoval tries to remove customers, which can easily be exchanged. It removes a set of customers based on how much they are alike. The closer they are compared to distance, amount of required goods, the time of day their respective routes visit them and whether they are on the same route or in the same region. The regions will be described in section 4.1. The five factors are added together with a percentage for each. The customers are sorted according to relatedness to a starting customer. In order to exchange customers not entirely alike, a randomness factor is used, with higher probability for customers more related to the starting customer.

$$R(i,j) = \varphi c_{i,j} + \chi|t_{ik_i} - t_{jk_j}| + \psi|d_i - d_j| + \omega T_{ij} + \eta A_{ij} \tag{3.1}$$

T_{ij} evaluates to 1 if the customers i and j are on the same vehicle before removed, and 0 otherwise. t_{ik_i} is the time vehicle k_i visits customer i, and k_i is the vehicle that visits customer i. A_{ij} is a variable set to 1 if the two customers are in the same region in the initial solution.

Algorithm 3.1 ShawRemoval

```
SHAWREMOVAL($s \in \{solutions\}, q \leq |C|, p \in \mathbb{R}_+$)
1  customer $c$ = a randomly selected customer from $s$
2  set of customers $D = \{r\}$
3  while $|D| < q$
4        do $r$ = a randomly selected customer from $D$
5           Array $L$ = an array containing all customers from $s \notin D$
6           sort $L$ such that $i < j \Leftarrow R(r, L[i]) < R(r, L[j])$
7           choose a random number $y$ from the interval [0,1)
8           $D = D \cup \{L[y^p|L|]\}$
9  remove the customers in $D$ from $s$
```

The values $c_{i,j}$, $|t_{ik_i} - t_{jk_j}|$ and $|d_i - d_j|$ are normalized as well as the factors $\varphi, \chi, \psi, \omega$ and η will be normalized so their sum is 1, in order to have $0 \leq R(i,j) \leq 1$.

Because the algorithm should move further around in the solution space, a randomness factor is used, this way the heuristic will not return the exact same solution every time the algorithm is run on a specific solution.

WorstRemoval
Worst removal removes the customers which add the highest cost to the solution value, meaning that a routes cost is decreased the most if this customer were removed from its current route.

Algorithm 3.2 WorstRemoval

```
WORSTREMOVAL($s \in \{solutions\}, q \leq |C|, p \in \mathbb{R}_+$)
1  while $q > 0$
2        do Array $L$ = all customers $i \in s$, sorted descending $cost(i, s)$
3           choose a random number $y$ from the interval [0,1)
4           customer $c = L[y^p|L|]$
5           remove $c$ from $s$
6           $q = q - 1$
```

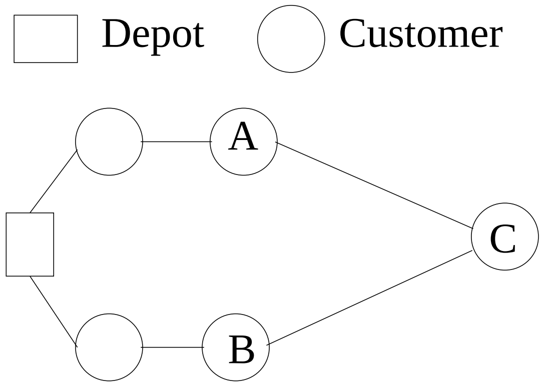
The cost of the customer i in the solution s is defined as: $cost(i,s) = f(s) - f_{-i}(s)$, where $f_{-i}(s)$ refers to the cost of solution s when customer i is removed.

The customers can be chosen in two ways, either the q worst placed customers in the current solution could be removed. Or in q iterations the worst placed customer could be removed, before recalculating the savings for the two neighbors of the removed customer.

In line 3 of algorithm 3.2 some randomness is added to the removal process, so the removing is not absolutely equal in each run of the program.

Two versions of the worst removal is created, one where the q worst customers from the solution are removed, without any recalculation between removals called WorstRemoval, and a version where the values for the customers is recalculated between each removal called Worst$_a$Removal. The difference between the to versions can be illustrated by the route in figure 3.1. The customer C is clearly the customer which would reduce the total length of the route most if removed. If recalculating, customers A and B would have their reduction value increased considerably, but otherwise they would probably not be removed.

Figure 3.1 Example of a route



Cleaning up
After the removal heuristic is run and before the removed customers is inserted, one or more empty routes possibly exists. Instead of using these as possible routes in the insertion heuristics, they are deleted, because it is always possible to create new routes if desired. If several empty routes were present, the insertion process could be troublesome.

3.2 Insertion heuristics

The insertion heuristics mentioned in section 2.1 are altered slightly to attend to the fact that the insertion heuristics no longer start with the empty set of routes, but start with a set of partly finished routes. The general idea is that the algorithms tries to put the removed customers into the old routes, but if that is not feasible, or starting a new route is cheaper, a new route is started. For the greedy algorithm, the memory using greedy algorithm and the regret algorithms the best place to put the customers in each of the partly routes is calculated, as well is the cost for starting a new route with the customer. From these is chosen which customer to insert and where to insert it using the same decision methods as the original algorithms.

In the sequential algorithm, each of the old routes is taken in some order and try to fill the chosen route, untill no more customers can be served by that route. When this happens I go to the next route. The order chosen is often the order they were created in the first place. When all routes have been tested the excess customers are used to create new routes according to the description in section 2.1.

The ALNS uses the following heuristics: simple greedy, memory using greedy, sequential greedy and regret-2 to regret-6, in all eight different insertion heuristics.

3.3 Adaptivity

Because the algorithm has 8 insertion heuristics and 4 removal heuristics, the program needs to choose which heuristics should be used in each iteration. To handle this choice a roulette wheel is introduced. The roulette wheel has a number of sectors corresponding to each of the heuristics. The size of the sectors can be different to accommodate different probabilities for each of the heuristics. The wheel is turned and chooses one insertion and one removal heuristic to be used in the current iteration.
When the algorithm runs many iterations, it would be wise to reflect the success of each heuristic in later iterations. This is handled by a adaptivity routine. After each iteration, the success is recorded by adding a bonus to a score s_i for each heuristic i. The bonuses are only awarded to the heuristics, if a previously unvisited solution is met, because the heuristics should not be encouraged to search the same part of the solution space again. To keep track of which solutions have been reached so far, a hash table is used. The

bonus categories can be seen in the table 3.1.

Parameter	Description
ϕ_1	The last remove-insert operation resulted in a new global best solution.
ϕ_2	The last remove-insert operation resulted in a solution that has not been accepted before, the cost of the solution is better than the cost of the current solution.
ϕ_3	The last remove-insert operation resulted in a solution that has not been accepted before, the cost of the solution is worse than the cost of the current solution.

Table 3.1: The bonuses for finding a new solution.

The ALNS algorithm looks at three parameters when assigning probabilities to the removal and reconstruction heuristics. ϕ_1 is awarded, when a new global best solution is discovered, this should be awarded most, because heuristics yielding the new best solutions are wanted. ϕ_2 and ϕ_3 are basically the same, as they reward heuristics ability to diversify the search. Both the removal and insertion heuristic is awarded the same bonus, as it is very difficult to distinguish which of the two were the main reason for success.

The iterations are divided into segments of a given size, after each segment the probabilities of the roulette wheel is changed according to the bonuses. Because the choice of heuristic is based on probabilities it can be wise to keep some information from the last probability set. Thus the new set of probabilities are calculated as follows. The probability of choosing heuristic i in segment j is denoted w_{ij}. After segment j the weights for segment$j+1$ is calculated as follows.

$$w_{ij+1} = w_{ij}(1-r) + r\frac{s_i}{a_i} \tag{3.2}$$

The score s_i is divided by the number of times heuristic i was chosen a_i and added to the weight of the last segment scaled by the reaction factor r. If $a_i = 0$ then it is set to 1, because s_i will also be 0 in that case. The size of r controls how fast the algorithm adapts to the new scores. The higher r is, the faster good heuristics will be chosen at higher probability. If r is zero, there is no adaptivity, and if r is one, no information from the last segment is kept.

In each iteration a number $5 \leq \delta \leq \min(100, \zeta n)$ of customers is removed, where ζ is a parameter to be tuned. 5 is chosen arbitrarily, in order to make some change in each iteration, and the maximum of 100 customers is chosen so the removal and insertion of customers is not too time consuming in order to quickly run several iterations.

3.4 Acceptance and stop criteria

The acceptance of a solution is decided using simulated annealing, but any metaheuristic could be used, by calculating the value $e^{-\frac{f(s')-f(s)}{T}}$, with $T > 0$. T is decreased in each iteration by a factor $0 < c < 1$. The starting temperature T_{start} is calculated after the first solution is generated, as proposed by Røpke and Pisinger [17]. The starting temperature is set so a solution that is w times worse than the first solution is accepted with a possibility of 0.5. which leads to $T_{start} = -\frac{w \cdot f(s)}{\ln(0.5)}$.

The factor w has to be tested as well as value c. Røpke and Pisinger finds the value 0.99975 to be a good for c.

The ALNS stops when a specific number of iterations have been run.

3.5 The overall algorithm

The descriptions in the previous sections lead to the overall algorithm shown in algorithm 3.3.

Algorithm 3.3 The overall algorithm

```
ALNS(customers)
1  make initial solution
2  while stop criteria not met
3       do choose algorithms
4          remove customers
5          insert customers
6          update scores
7  return best solution
```

Chapter 4

Initial solution

In this chapter I will describe the method I propose to use for the creation of the initial solution used as a base for the ALNS algorithm described in chapter 3. I will also describe the real world motivation for using this method, and some different versions of the method.

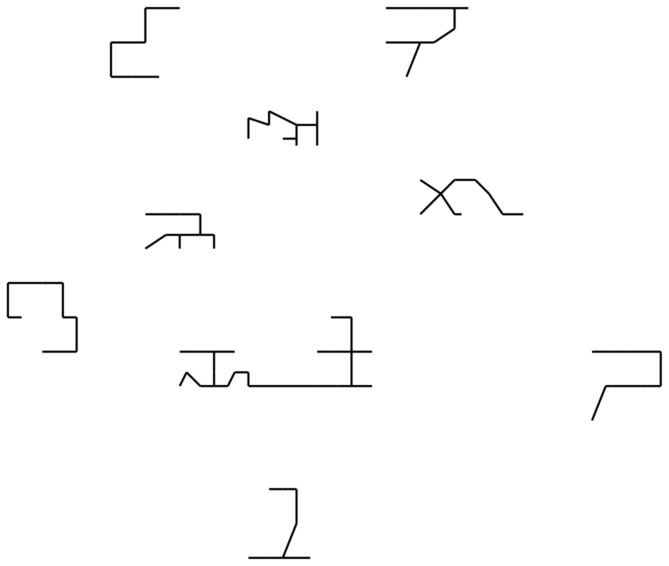
The method is basically a Divide and Conquer scheme. I will divide the data according to some relations, these will be described and discussed in section 4.1 and motivated in section 4.2. I will describe the chosen solution used to divide the problem in section 4.3, and go into detail about the algorithm used in section 4.4. In section 4.5 I will discuss the method used to solve the VRPTW in the regions, and describe an optimization method which could be used before giving the solution to the ALNS algorithm. In section 4.6 I will test the insertion heuristics described in section 2.1 on some of the test instances by Solomon to see how they perform on the divided problems compared to the undivided problems.

4.1 The division

When dividing the problem into subproblems it will probably be a good idea to consider the relatedness of the customers. The relations between the customers could be location, traveling distance or travel time between customers or it could be the time window constraints. If the traveling distance between customers is considered, the minimum spanning trees algorithms could be used to group the customers.

If a forest of minimum spanning trees is build, these trees could be the subproblems to solve. To test the theory, forests consisting of ten trees are build using Kruskal's Algorithm which is explained in algorithm 4.1 on page 32. The forests are created for some of the original Solomon instances, one in each category. These forests are shown in figures 4.1, 4.2 and 4.3 on pages 26-27. I have only drawn the arcs used in the tree, as a result, trees consisting of only one node cannot be seen in the figures.

Figure 4.1 The minimum spanning forest for Solomons c101 instance, with 10 trees



As figure 4.1, 4.2 and 4.3 shows, the trees are very different in size, this could be accounted for by adding a constraint on the size of the trees, but this would mean that the trees would not be the minimum spanning trees. Therefore they would not describe the most closely related subdivisions.
Besides being different in size, the large tree in figures 4.2 and 4.3 have nodes all over the plane, and 7 of the trees in figure 4.2 consist of only one customer, this would mean that one of the subproblems would not be much smaller than the original problem. In these instances this kind of division scheme would not give much.

Figure 4.2 The minimum spanning forest for Solomons r101 instance, with 10 trees

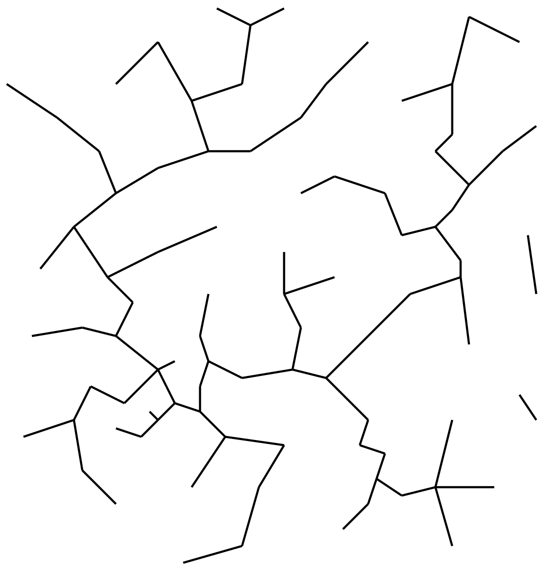
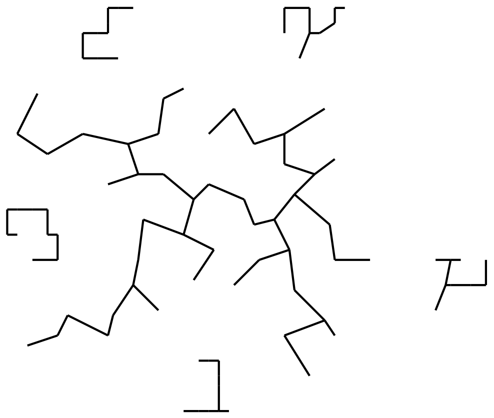


Figure 4.3 The minimum spanning forest for Solomons rc101 instance, with 10 trees



If the problem is divided into subproblems according to the coordinates of the customers, a sweep line algorithm with the sweep line rooted at the depot could be used. This algorithm is described by Gillett and Miller [28] for the VRP, it is known as a cluster first route second algorithm. Because this algorithm is for the VRP, it does not account for the Time windows. But if the problem is parted into subdivisions that could contain more than one route, this could be a division strategy.

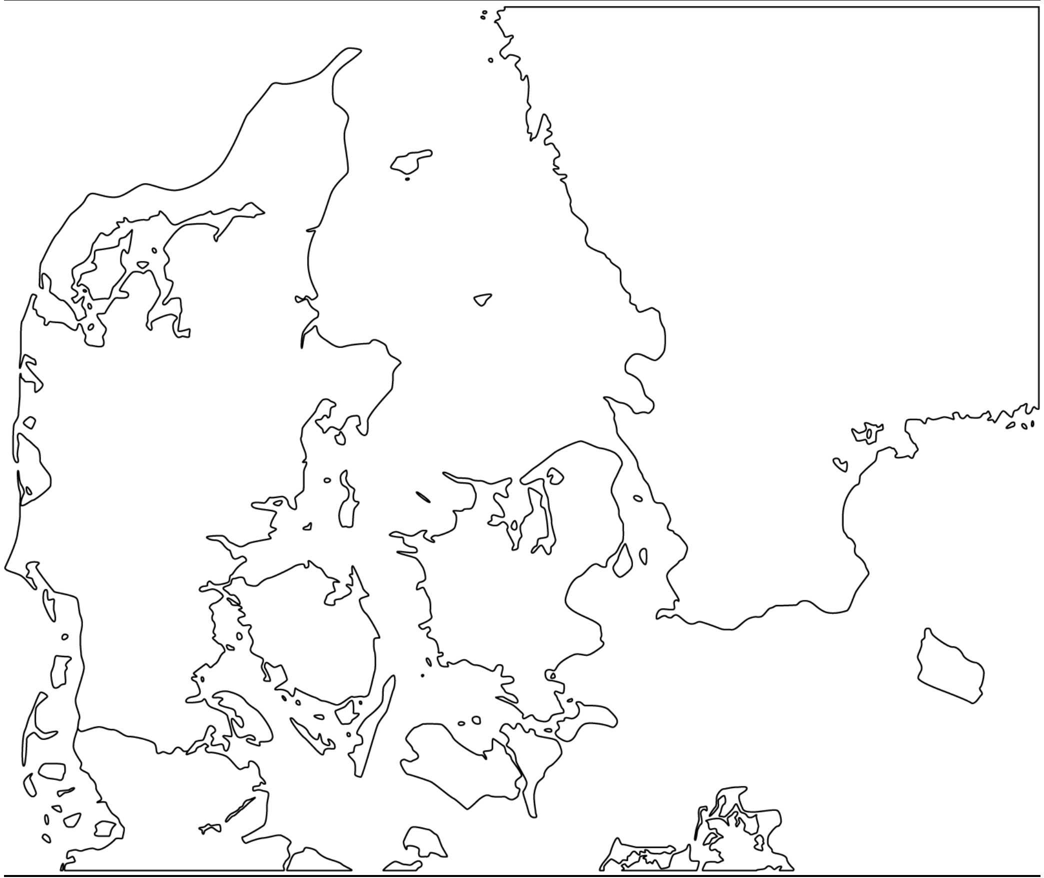
Another way could be to divide the plane into some regions not described by the angle of the customer from the depot, but by relation to some points in the plane. Voronoi Diagrams could be used to divide the plane into regions, and in these regions solve the VRPTW. I will describe the Voronoi Diagrams in detail in section 4.3, but first I will give a geografical argument for doing so.

4.2 The geography

Considering the geography of Denmark, with plenty of islands and peninsulas, and a good deal of bridges combining all these parts. The country of Denmark can be viewed in figure 4.4. If a route were to cross one of these long bridges more than twice during a day the driver would probably think it was not a good route. Instead of treating the entire country as a region in which to solve the VRP, the country can be split up in regions such as the major islands, and the large peninsula in one or two regions.

As it was the case with the ShawRemoval algorithm from section 3.1, it is preferred to have customers on the same small island to be in the same region if possible. If information about the actual geography of the problem is provided, the problem can be divided according to the geography. But as most problems do not give such information in the data, another way to divide cleverly has to be found.

Figure 4.4 The country of Denmark

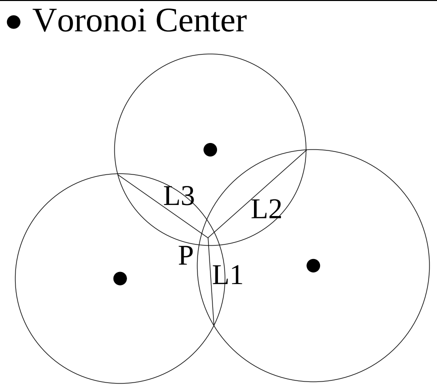


4.3 Voronoi Diagrams

The Voronoi Diagrams can be used to describe the plane as regions where each point in that region is closer to the defining point of the region, than to any other defining points of the diagram. The defining points are called Voronoi Centers, as they are centers of the circles used to generate the region borders.

The Voronoi Diagram can be produced by centering circles at each Voronoi center and increase the radii. When two circles meet they produce a line segment until the meet a third (or maybe more) circles at which point they produce a point, and create new line segments with the new circles, an illustration of three circles interacting is shown in figure 4.5. The construction continues until there is only unbounded circles left.

Figure 4.5 Illustration of Voronoi generation



The figure illustrates three circles centered at Voronoi Centers. The lines L1, L2 and L3 are produced as the circles meet. The point P is the point where the three lines meet.

A Voronoi Diagram is a planar graph with some useful properties, the regions of the Voronoi Diagram are the same as the faces of the planar graph. The number of edges (n_e) and vertices(n_v) in the Voronoi Diagram are connected to the number of Voronoi centers (n_c). As all other planar graphs the relationship between the edges (m_e), the vertices (m_v) and the faces (m_f) of the graph is.

$$m_v - m_e + m_f = 2 \tag{4.1}$$

The number of faces in the Voronoi Diagram is equal to the number of Voronoi centers. Because the half-edges on the border of the Voronoi Diagram have no vertices in one end, the vertice v_∞ is set to be the end of all half-edges. which leeds to this equation for the Voronoi Diagram.

$$(n_v + 1) - n_e + n_c = 2 \tag{4.2}$$

The Vertices in the Voronoi Diagram all have degree at least 3, because there needs to be at least three circles centered at a Voronoi Center to produce a Voronoi node. When the vertice v_∞ is also considered each edge connects exactly 2 vertices, summing up all the edges from each vertice the following relationship is reached.

$$2n_e \geq 3(n_v + 1) \tag{4.3}$$

.

This give the following calculations.

$$\begin{aligned} (n_v + 1) - n_e + n_c &= 2 \qquad (4.4) \\ n_e &\leq 3n_c - 6 \qquad (4.5) \end{aligned}$$

This means that there is at most $3n_c - 6$ pairs of regions with borders to each other. The number of vertices can be bounded in the same way to $n_v \leq 2n_c - 5$

Geography
In order to divide the problem into subproblems using Voronoi Diagrams, one has to find some centers of the regions. If the problem at hand is a Multi Depot VRP, the different depots are a good basis for the division centers. If not other ways to find the center must be found.

When dealing with a problem where the customers are represented as points in the Euclidean plane, Kruskal's algorithm for minimum spanning trees could be used regarding the edge weights, and when there are only N parts the centers of mass in these could be found and use as the centers for the Voronoi Diagrams. It could be possible that the N parts could be a good division of the problem into subproblems, if the Voronoi division does not defer much from the N parts.

Another possibility is to find the pair of customers with the minimal distance and take the point in the middle of these and remove the two original points. If one continues in this way until there are only N points left these points could be used as the Voronoi centers.

The distance used in the division algorithms is mostly the Euclidean distance between the points, but in real world data, the distance will also be the actual distance, or even the transportation time between two customers.

Algorithm 4.1 Kruskal division

```
KRUSKAL(graph $G(V, E)$)
 1  $num\_trees \leftarrow Num\_customers$
 2  $sort(E)$▷ by length
 3  while $num\_trees > divisions$
 4        do $e1 = take\_min(E)$
 5           if ends(e1) is combined
 6              then remove e1
 7              else  add e1 to tree and $num\_trees$- -
 8  for $i = 0 \rightarrow divisions$
 9  if size(treei) small enough
10      then continue
11      else  $edges \leftarrow treei.edges$,
12            $divisions \leftarrow size(treei)/wished\_size$,
13            goto 2
```

The Kruskal algorithm 4.1 creates a minimum spanning forest on the edge weights. The while loop from line 3 to 7 adds the smallest unused edge to the forest, as long as the introduction of the edge does not create a cycle in a tree. The while loop ends when the wished number of trees are created. The for statement in line 8 ensures that the trees are not to large. For instance if a tree has more than $\frac{2}{3}$ of the total number of edges, then the division is probably not good. As figure 4.2 shows.

4.4 Voronoi Diagram Algorithm

The Voronoi Diagram algorithm solves the problem of finding the regions closest to a set of given points in the plane. For instance if a company wants to figure out if a placement of a convenience store is good or bad, the company can run the Voronoi Diagram Algorithm with the old stores, and the proposed site for the new store. The output will be a graph representing the regions covered by the stores, and if the new store has a customer base which is too small the placement is probably not good. On the other hand if the company runs the algorithm with only the old stores, regions that are large would indicate where to place a new store.

The algorithm is a so called sweep line algorithm, in which a line sweeps the entire plane from top to bottom and keep a list of events still to happen. When these events happen they are handled and new events may be added to the event list. The algorithm is shown in algorithm 4.2
The algorithm takes a set of points (x_i, y_i) in the plane as input, these points are referred to as sites or Voronoi centers. The output is a set of points and line segments describing a division of the plane in regions around the sites. Above the sweep line is a beach line with a number of parabola sections. The parabolas symbolize the points which are half the distance between the Voronoi centers and the sweep line. The beach line is kept updated when some events occur. Where two parabola sections meet they make a point in the output, and as the parabolas increase in size the points become lines in the output, these are the borders of the regions, an illustration of the sweep line and the beach line is shown in figure 4.6.

The algorithm scans the plane from top to bottom. Each time an event occurs, the event is handled. The events come in two classes, Site events and Circle events. See figure 4.7 for an illustration of the site event.

Site events happen when the sweep line meets a previously unhandled site. If two sites have the same y value they are handled according to x value. The site is handled by adding a new parabola to the beach line and the circle events for this new parabola are calculated and inserted into the priority queue.

Circle events happen when a parabola section disappears because its two neighboring parabola sections completely overlap it. When this happens the point where the parabola section disappears is registered in the output, and the lines the parabola section was creating are stopped at this point. It is called a circle event, because the original method was to expand circles until they met.

The implementation of algorithm 4.2 is taken from the c++ version (O'Sullivan [29]) of Fortune [30].

Algorithm 4.2 Voronoi Algorithm from de Berg et al. [20]

```
VORONOI(A set $P = \{p_1, \dots, p_n\}$ of points in the plane)
1  $Q \leftarrow$ site events $\triangleright$ Priority Queue
2  $T \leftarrow \emptyset$ $\triangleright$ status structure
3  $D \leftarrow \emptyset$ $\triangleright$ return structure
4  while $Q \neq \emptyset$
5      do Remove the event with largest $y$coordinate from $Q$
6         if the event is a site event occurring at site $p_i$
7            then HandleSiteEvent ($p_i$)
8            else HandleCircleEvent ($\gamma$)
9  bind the half edges, so they are bounded within a bounding box
10 return the line segments of the Voronoi Diagram
```

Other center finding algorithms
In stead of trying to find the Voronoi centers by looking at the data, it is possible to divide the bounding box into an n by n grid, using the lines $y = xmin + \frac{i}{n} \cdot \Delta x$ and $x = ymin + \frac{i}{n} \cdot \Delta y$, where $i = 1, \dots, n-1$. It is easy to divide the customers by these lines, which means that it is not necessary to run the Voronoi algorithm with this division.

The Voronoi algorithm with different center finding algorithms and the n by n divisions of the plane all have the advantage that they produce neighboring regions, which makes it easy to find possible good routes to combine or exchange from.

Datastructures
When a Voronoi Diagram is found, it is necessary to compute which region each customer belongs to. This can be done with a trapezoidal map as described by de Berg et al. [20]. The trapezoidal map is a general data

Figure 4.6 Illustration of the sweep line and beach line

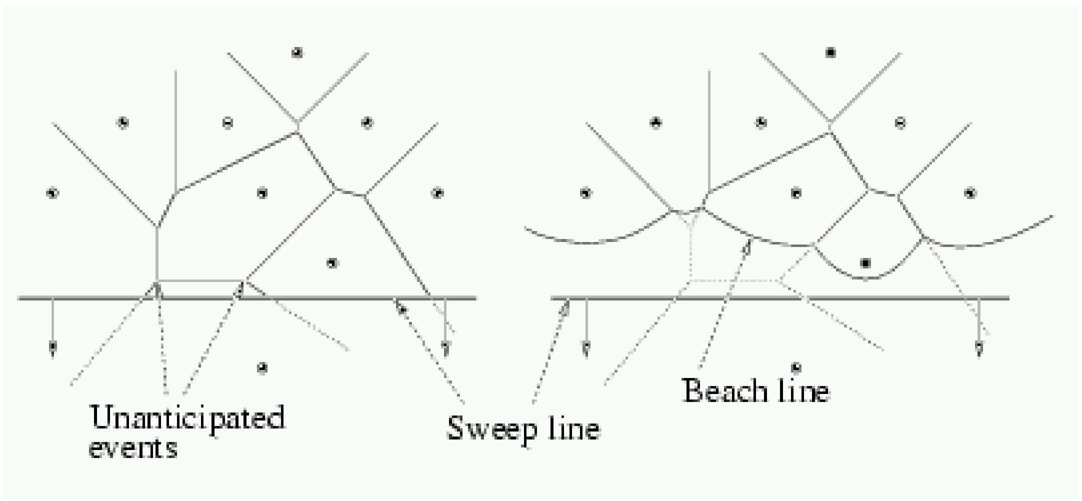
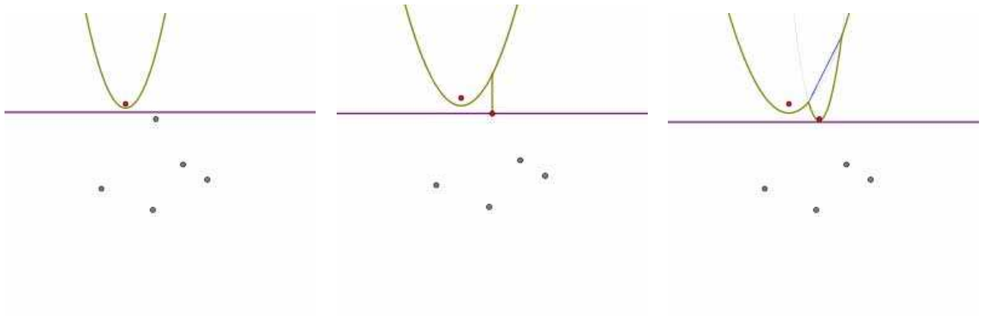


Figure 4.7 Illustration of the Site event



(a) Before meeting the site (b) Meeting the site (c) After meeting the site

When a site event occurs a straight line is placed from the site vertically to the beachline (b). When the event has occurred the parabola of the site is increased yielding a straight line (c).

structure, which is very useful when locating points in a graph that is not necessarily connected. When looking up a point, it will return the trapezoid in which the point is located. The trapezoid is located inside a region of the graph. But when dealing with a connected planar graph as the Voronoi Diagram is if the bounding box is large enough, a data structure which is simpler to compute can be used. This data structure shall return the region the point is located in, and not a part of a region when looking up. I call this data structure a TZTree, because it have a basic inner structure which is a tree, and it is build in a way similar to the trapezoidal map. The TZTree is a treelike structure, with some of the same characteristics of a binary search tree. The major difference is, a leaf does not have exactly one way down to it in the structure. How the TZTree looks and how it corresponds to a Voronoi Diagram is described in example 4.1

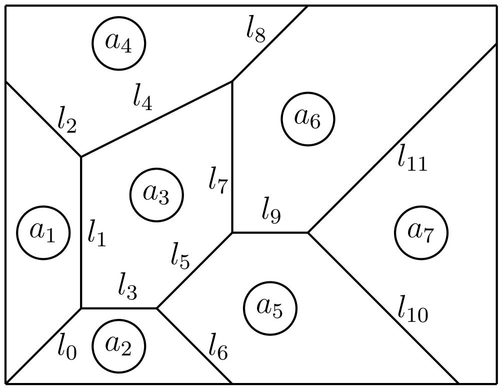
Building the TZTree
When building the TZTree as shown in example 4.1, a line segment with one of its points located on the bounding box has to be found and inserted. When inserting a new line segment l the algorithm finds the line segment l_1 in the TZTree which endpoint p is one of the endpoints p_1 of l. If the other end point p_2 is to the left of l_1 then l is inserted as the left child in the TZTree. The insertion continues in this way until no more line segments are left uninserted. If a line segment cannot be placed at the current time, it is saved and inserted later. The algorithm is shown in algorithm 4.3

Searching for a region using the TZTree
When locating the area A where the point p is located, p is compared to the line segment in the root node. The comparison is a cross product which will tell whether the point is to the left or to the right of the line segment. The result of the comparison is used to traverse the TZTree, until the node has no children, thus the area where the point is located is found.

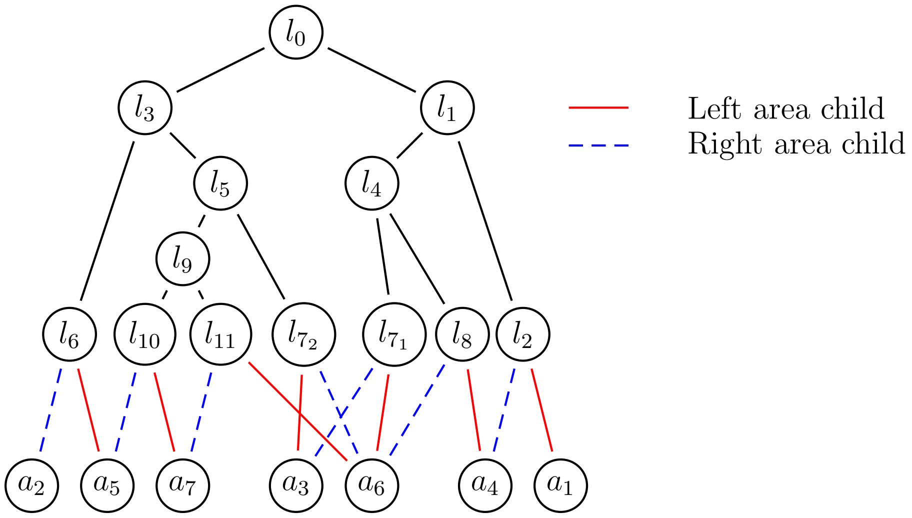
When a line segment on the border of the Voronoi Diagram, or an edge that crosses the original line segment is reached, the points location is known.

If there are 4 or more line segments meeting at a given point in the Voronoi Diagram, the TZTree will not work, in these cases one of the Voronoi centers could be moved a bit, which would give a slightly different division of the customers, but this is preferable to having to take the special case into consideration.

Example 4.1 TZTree



The Voronoi Diagram can be represented as a TZTree as shown below.



The areas at the bottom line represent the areas in the Voronoi Diagram at the top of the example

Algorithm 4.3 Building TZTree

```
TZTree(a list of line segments: lList)
 1  TZTree Tree ← ∅
 2  while lList ≠ ∅
 3        do choose line segment ls1
 4           find tree node t1 containing ls1.p1 or ls1.p2
 5           decide if ls1 is left or right of t1
 6           insert ls1 as appropriate child
 7           if area is closed
 8              then create new area
 9              else  set area
10           update corners
11  update neighbors
```

4.5 Solving the local VRPTW

The local VRPTW algorithm could either be an exact algorithm in order to get the partially solutions best possible, or it could be heuristic. When it is considered that the algorithm to divide the problem into subproblems is a heuristic, and that a heuristic is used to assemble the solutions of the subproblems into a final solution. It could be a good idea to solve the local VRPTW problems with a heuristic, in order to have time to make many iterations to diversify the searching of solutions. Such heuristics could be some of the same as the ones used in the ALNS phase of the algorithm.

The advantage of the exact algorithm would be that the subsolutions would be the best possible for the subproblems, and thus should make a good starting point for assembling the routes that are not entirely filled in this part of the algorithm. This could work fairly well if the subproblems are small enough, and hence not too time consuming to solve.

Optimizing

The routes generated for each region, could be optimized by exchanging some customers from one of the routes to one of the other routes if more than one route is created in a region. This optimization could be done by running the ALNS algorithm on the individual regions or by using some of the optimization heuristics mentioned in section 2.2.

When the subproblems have been solved there will most likely be some solutions with vehicles that are no more than half full. These half filled vehicles can be combined if they are not far from each other. If one half filled vehicle from one region is in the vicinity of another half filled vehicle from another region, then these can be combined by adding the goods and customers from one of the vehicles to the other. If there are to many goods, or if the time windows becomes a problem, the excess customers can be tried to put into the other vehicles from the same region or from other regions, if possible.

4.6 Testing the division

To see how well the division is compared to the undivided solution, I have run the initial route creation heuristics on some instances. Once with the Voronoi Diagram created with centers found by the Kruskal algorithm, and once with undivided problems. The results are shown in appendix A. The problem data chosen for the test are from the Solomon [4] and Homberger [5] test instances. In all three Solomon categories the instances 101, 105, 201 and 205 are chosen. From Homberger [5] the chosen test instances are C1_2_1, C1_2_8, C2_2_1 and C2_2_8. These 16 instances have been chosen as representatives because they are from different halves of the individual categories. The test instances have been grouped in pairs for each of the categories in tables 4.1-4.4.

In the tables the gap is calculated as:

$$\frac{sol_{no-div} - sol_{Kruskal}}{sol_{no-div}} \tag{4.6}$$

Table 4.1: Divided vs non divided initial solution

Simple greedy insertion algorithm		Memory greedy insertion algorithm	
Instance	improvement	Instance	improvement
c10x	6%	c10x	6%
c20x	53%	c20x	53%
C1_2_X	4%	C1_2_X	4%
C2_2_X	-3%	C2_2_X	-4%
r10x	-2%	r10x	-4%
r20x	-3%	r20x	-4%
rc10x	-14%	rc10x	-14%
rc20x	-4%	rc20x	-4%
Avg gap	5%	Avg gap	4%

In table 4.1 the two greedy insertion heuristic are tested, they give almost the same solutions, but the basic greedy heuristic does slightly better on the randomized instances. They have the second and third best result of all the heruistic, therefore they are both kept.

Table 4.2: Divided vs non-divided initial solution 2

regret-2 insertion algorithm		regret-3 insertion algorithm	
Instance	improvement	Instance	improvement
c10x	5%	c10x	20%
c20x	-11%	c20x	-67%
C1_2_X	23%	C1_2_X	17%
C2_2_X	16%	C2_2_X	-3%
r10x	0%	r10x	-1%
r20x	15%	r20x	-13%
rc10x	5%	rc10x	0%
rc20x	22%	rc20x	-8%
Avg gap	15%	Avg gap	-10%

Tables 4.1-4.4, show how improved the solutions are when the dataset is divided using Voronoi Diagrams with the centers found by the Kruskal Algorithm compared to the solutions when the datasets was not divided. Better solutions are shown with a positive percentage and worse solutions are shown with negative percentages. For all the insertion algorithms, the average is also reported.

Table 4.3: Divided vs non-divided initial solution 3

regret-4 insertion algorithm		regret-5 insertion algorithm	
Instance	improvement	Instance	improvement
c10x	24%	c10x	29%
c20x	-47%	c20x	-6%
C1_2_X	17%	b C1_2_X	18%
C2_2_X	-36%	C2_2_X	2%
r10x	-5%	r10x	-6%
r20x	-14%	r20x	-8%
rc10x	-4%	rc10x	-8%
rc20x	6%	rc20x	4%
Avg gap	-7%	Avg gap	3%

Table 4.4: Divided vs non-divided initial solution 4

regret-6 insertion algorithm		sequential insertion algorithm	
Instance	improvement	Instance	improvement
c10x	31%	c10x	7%
c20x	-8%	c20x	-43%
C1_2_X	2%	C1_2_X	18%
C2_2_X	-12%	C2_2_X	-28%
r10x	-5%	r10x	-2%
r20x	-10%	r20x	-4%
rc10x	-1%	rc10x	-2%
rc20x	2%	rc20x	-10%
Avg gap	2%	Avg gap	-8%

As the tables show, the only group that does not show improvements with any of the insertion heuristics is r10x, and in fact of those two it is only r101 which does not show any improvement at all. This means that the division of the problem into sub problems can create better initial solution for the majority of the instances tested. For all the insertion heuristics, the clustered instances are improved. The c101 instance is actually solved to optimality by the simple greedy, the memory using greedy and the sequential greedy algorithms, when divided.

Chapter 5

Testing

In this chapter I will tune some of the parameters described in chapters 2 and 3; this tuning is described in section 5.1. When the program has been tuned I will run the program on all the instances provided by Solomon [4] and Homberger [5], this is done in section 5.2. I will compare the results to the best known solutions. The hope is that every instance can be solved quickly, i.e. in less than 5 minutes, and that the results are not far worse than the previously best known results. It is assumed that the algorithm will have the best results on those instances which are clustered because the algorithm takes advantage of their structure.

I will also run the program on datasets provided by a Danish route-planning company. These instances will be tested with the Euclidean distance as well as the actual distance between customers, the time used to traverse the edges of the graph will be the actual time needed, as well as the time used when traveling at 65 km/h.

The test are done on a Intel(R) Pentium(R) 4 CPU 3.00GHz processor with 2GB Memory.

5.1 Parameter tuning

For the parameter tuning of the ALNS algorithm, a subset of the instances by Solomon [4] have been used. The set consists of two instances from each group of instances. As the numbers in Solomons instances refer to how they are structured, the instances have been chosen to represent the diversity. The chosen instances are: c101, c105, c201, c205, r101, r105, r201, r205, rc101, rc105, rc201 and rc205. For every test run the maximum number of iterations is 25000.

Because the algorithm is a heuristic there are no guarantees of how good the results will be. Therefore every test for the algorithm is run several times with each setting, in order to smooth out really bad test runs. The gap between the result and the optimal solution is calculated as well as the average of these. The gap g_{ijs} for instance i in test run j with parameter setting s is calculated using the optimal solution opt_i and the found solution sol_{ijs}.

$$g_{ijs} = \frac{sol_{ijs} - opt_i}{opt_i} \tag{5.1}$$

Since the algorithm is run several times for each instance the average gap is calculated and because the algorithm is not deterministic it will also be reasonable to regard the best result for each instance min_{is}.

$$min_{is} = \min_j g_{ijs} \tag{5.2}$$

In order to test the best possible solutions made, the minimum gap for each of the instances is found as well as the maximum of those. I call this the mm_s gap, this gap show how bad the best result can be.

$$mm_s = \max_i min_{is} \tag{5.3}$$

The average gap for each setting is calculated as the sum of the results for all the test runs for all the instances divided by the total number of tests. This number describes how well the setting was on average, whereas the mm_s

indicates how good the best solution can be expected to be. This means that the mm_s for a setting can be smaller than the average gap for that setting. These two measurements are chosen in order to see how well a setting is on average, and to see how well it is reasonable to assume the setting can be.

Tuning parameters
The parameters mentioned in chapter 2 and 3 are: ϕ_1, ϕ_2 and ϕ_3 describing the bonus for accepting a new solution, c and w controlling the descent speed of the temperature and the starting temperature. p_{worst}, p_{worst_a} and p_{shaw} controlling the randomness of the removal heuristics must be tested, as well as the parameters controlling the ShawRemoval $\varphi, \chi, \psi, \omega$ and η. ζ controlling the maximum number of customers removed in each iteration and r controlling how fast the ALNS adapts to the results of the heuristics must also be tested.

In order to tune the parameters which are specific to the algorithm described in this thesis, I have chosen to use some of the values Røpke and Pisinger [17] found to be good for their version of the ALNS. These are shown in table 5.1.

ϕ_1	ϕ_2	ϕ_3	c
33	9	13	0.99975

Table 5.1: The parameters used in ALNS which are not tested.

Tuning the removal heuristics randomness
The three removal heuristics ShawRemoval, WorstRemoval and Worst$_a$Removal have the parameters p_{worst}, p_{worst_a} and p_{shaw} which ensures that the randomness of these heuristics is not too dominant, in order to make good removals. To tune these a series of tests is run and the average gap as well as the mm_s gap is reported. The results of the tests are shown in tables 5.2-5.4. The parameters tested are chosen to be $2-6$, if $p = 1$ the removal heuristic will be totally random.

p_{worst}	2	3	4	5	6
Avg gap	23%	23%	**22%**	24%	22%
max min gap	41%	30%	**30%**	50%	38%

Table 5.2: Testing p_{worst}

p_{worst_a}	2	3	4	5	6
Avg gap	9%	8%	8%	8%	8%
max min gap	11%	12%	11%	**6%**	7%

Table 5.3: Testing p_{worst_a}

p_{shaw}	2	3	4	5	6
Avg gap	**9%**	11%	11%	11%	11%
max min gap	12%	15%	14%	18%	15%

Table 5.4: Testing p_{shaw}

From these tests I conclude that $p_{worst} = 4$ and $p_{worst_a} = 5$ are good values. p_{shaw} is best at 2, although this indicates that the ShawRemoval is rather random in its selection of customers to remove from the routes. It is also shown in these tables that the first version of WorstRemoval is a lot worse than the other two heuristics, but in order to get a diversity in the ALNS search, it is kept.

Tuning the ShawRemoval parameters

The relation between the factors $\varphi, \chi, \psi, \omega$ and η in the ShawRemoval described in equation (3.1) on page 19 are tested. To minimize the testing I have used the relations between φ, χ, ψ and ω from Røpke and Pisinger [17], as they have done extensive testing on these. The tests done here are merely to see if the regions proposed in this thesis have any impact at all.I will only test different values of ω and η. The other values are set to $\varphi = 9, \chi = 3$ and $\psi = 2$ for all the tests. The tested values for ω and η are shown in table 5.5, together with the average gap and the mm_s gap.

parameter	ω	η	avg gap	mm_s gap
s1	0	0	7 %	12 %
s2	0	3	7 %	14 %
s3	0	6	7 %	12 %
s4	3	0	6 %	12 %
s5	3	3	7 %	12 %
s6	3	6	6 %	**5** %
s7	6	0	6 %	12 %
s8	6	3	6 %	12 %
s9	6	6	7 %	7 %

Table 5.5: Testing the shaw parameters

From these results I conclude that setting s6 should be used because the average gap of the tested parameter settings is not very different, and the mm_s gap of setting s6 is better than all the other results. This setting implies that the regions two customers belong to is twice as important as the routes of the customers.

The choice of setting s6 indicates that the regions used in the Voronoi division does have some effect on the ShawRemoval, and is more important then the routes when removing customers.

Tuning the ALNS algorithm

In tables 5.6-5.8, the different values of ζ, w and r are tested. The lines in the tables show the values tested, the average gap between the results of the test and the optimal solution and the final row shows the mm_s gap.

ζ	10%	20%	30%	35%	40%	45%
Avg gap	8%	8%	8%	8%	8%	8%
max min gap	11%	12%	**9%**	10%	10%	10%

Table 5.6: Testing removal percentages

w	5%	10%	20%	30%	35%	40%	45%
Avg gap	4%	3%	3%	3%	3%	3%	3%
max min gap	7%	7%	7%	6%	7%	**4%**	6%

Table 5.7: Testing the starting temperature

r	0.10	0.25	0.4	0.5	0.6	0.75	0.9
Avg gap	**4%**	4%	5%	5%	5%	5%	5%
max min gap	6%	6%	6%	7%	7%	6%	7%

Table 5.8: Testing the reaction factor

From these three tables I conclude that ζ should be 30%, this means that in every iteration it is possible to explore a rather large neighborhood. I also conclude that w should be 40% which implies that the ALNS will accept solutions which ar far worse than the starting solution in order to broaden the search at the beginning. Finally r should be 0.10 indicating that the adaptivity of the algorithm is slow. This means that the parameters for the final tests will be:

ϕ_1	ϕ_2	ϕ_3	c	w
33	9	13	0.99975	0.4
p_{worst}	p_{worst_a}	p_{shaw}	ζ	r
4	5	2	0.3	0.1
φ	χ	ψ	ω	η
9	3	2	3	6

Table 5.9: The final parameters used in ALNS

5.2 VRPTW Testing

After tuning the parameters, a series of tests are done on all the available test instances using the parameters from table 5.9. These tests are the focus of the remainder of the chapter.

The testing is done on the Solomon [4] instances as well as the extended Solomon instances by Homberger [5] for comparison of the heuristic to earlier solution methods. The Solomon instances are grouped in 3 categories C1 and C2, R1 and R2, and RC1 and RC2. The C* instances have the customers clustered, in the R* instances the customers are randomly distributed in the plane and the RC* instances have both clustered and randomly distributed customers. Each of the 3 categories have 2 classes, class 1 has a vehicle capacity of 200 and class 2 has a vehicle capacity of 700.
The original instances have 100 customers per problem, the extended instances have 200 - 1000 customers.

In order to see if the heuristic will do well on real world data, some sets of data have been provided from a route planning company. Unfortunately I do not have any previous results for the real world data, but I will test them using different distances between customers, and different time usage computations. These tests can tell if there is a significant difference when using the real world distance and the Euclidean distance.
The data is distributed over 5 days with up to 250 customers each day, in total there are 1101 customers. The problem instances did not have time windows when I got them, so the time windows for the real world data are randomly generated.

Solomon Results
Table 5.10 show the best results of the algorithm run on the clustered instances by Solomon. For almost all of the instances have the optimal solution is found at least once, and all the tests have used less than 15 seconds to find the results.

In table 5.11 the best results for the randomly distributed instances are shown. All results are within 9% from their optimum, and all three of the instances previously not solved to optimality have been solved to better values than the ones reported by Solomon [4]. Unfortunately the heuristic solutions reported by Solomon [4] have the objective of finding the solution using the fewest vehicles whereas the objective used in this thesis was to minimize the total length of the routes, thus it is not entirely comparable to my results. The routes for the better solutions are shown in appendix B

Table 5.12 contains my results for the partly clustered and partly randomized instances by Solomon [4]. The maximum gap between my results and the

Table 5.10: Results for the original clustered Solomon instances.

instance	result	# Vehicles	time (s)	o/h	best res	# Vehicles
c101.100	827.3	10	7.325	o	827.3	10
c102.100	827.3	10	7.517	o	827.3	10
c103.100	826.3	10	8.093	o	826.3	10
c104.100	844.6	10	9.159	o	822.9	10
c105.100	827.3	10	6.772	o	827.3	10
c106.100	827.3	10	7.909	o	827.3	10
c107.100	827.3	10	7.007	o	827.3	10
c108.100	827.3	10	8.007	o	827.3	10
c109.100	827.3	10	8.263	o	827.3	10
c201.100	627	4	8.354	o	589.1	3
c202.100	589.1	3	9.203	o	589.1	3
c203.100	588.7	3	10.633	o	588.7	3
c204.100	588.1	3	13.46	o	588.1	3
c205.100	586.4	3	8.596	o	586.4	3
c206.100	586	3	9.779	o	586.0	3
c207.100	585.8	3	9.85	o	585.8	3
c208.100	585.8	3	10.317	o	585.8	3

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions, the third last column indicates if the best known solution is optimal (o) or heuristic (h).

optimal solutions is 5%, and all the tests have running times less than 30 seconds. The rc206 instance has been solved to a better solution than the one reported by Solomon [4], but again the objective for that heuristic was to minimize the number of vehicles and not the total length of the routes.

Homberger Results
The best results for the Homberger instances are shown in appendix C, in table 5.13 a list of those results which are better than the results reported by Sintef [31]. The results on Sintef [31] are from heuristics with an objective of minimizing the number of vehicles, but I minimize the total length of the routes like the optimal algorithms do. Because the objective of the two solution methods are different the comparison between these results may seem strange, but since it is the only results I can compare my results to I

Table 5.11: Results for the original randomized Solomon instances.

instance	result	# Vehicles	time (s)	o/h	best res	# Vehicles
r101.100	1682.2	20	9.753	o	1637.7	20
r102.100	1489.4	18	9.671	o	1466.6	18
r103.100	1223	14	9.546	o	1208.7	14
r104.100	987.8	11	9.815	o	971.5	11
r105.100	1360	16	8.48	o	1355.3	15
r106.100	1248.9	13	18.481	o	1234.5	13
r107.100	1073.7	11	9.377	o	1064.6	11
r108.100	950.6	10	9.776	o	932.1	10
r109.100	1160.6	13	9.08	o	1146.9	13
r110.100	1093.8	12	19.192	o	1068	12
r111.100	1059.6	12	9.342	o	1048.7	12
r112.100	965	10	10.37	o	948.6	10
r201.100	1161.1	8	9.229	o	1143.2	8
r202.100	1052.6	**6**	9.662	o	1029.6	8
r203.100	900.6	6	11.186	o	870.8	6
r204.100	**752.1**	4	27.76	h	825.52	2
r205.100	978.2	5	19.924	o	949.8	5
r206.100	903.3	**4**	11.173	o	875.9	5
r207.100	865.8	**3**	13.591	o	794	4
r208.100	**725.1**	4	31.886	h	726.75	2
r209.100	877	**6**	10.888	o	854.8	9
r210.100	945.8	**4**	10.804	o	900.5	6
r211.100	**782.9**	4	31.028	h	892.71	2

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions, the third last column indicates if the best known solution is optimal (o) or heuristic (h).

use them. The routes created for the instances in table 5.13 are shown in appendix B.5-B.13.

The data for Hombergers R* and RC* with 800 customers was not able to be divided by the Kruskal and Voronoi method. The reason for this could be that the TZTree data structure is not as good as hoped. Therefore these are only tested using non-divided datasets.

Table 5.12: Results for the original clustered and randomized Solomon instances.

instance	result	# Vehicles	time (s)	o/h	best res	# Vehicles
rc101.100	1655.1	16	9.337	o	1619.8	15
rc102.100	1486.5	15	9.164	o	1457.4	14
rc103.100	1321.2	12	8.981	o	1258	11
rc104.100	1147.3	10	9.037	o	1132.3	10
rc105.100	1565.1	17	9.56	o	1513.7	15
rc106.100	1385.3	13	8.992	o	1372.7	13
rc107.100	1246.8	12	9.095	o	1207.8	12
rc108.100	1138.7	11	9.252	o	1114.2	11
rc201.100	1271.6	9	9.223	o	1261.8	9
rc202.100	1114.9	8	9.883	o	1092.3	8
rc203.100	946.9	5	11.027	o	923.7	5
rc204.100	814.6	4	13.68	h	798.41	3
rc205.100	1172.8	8	9.699	o	1154	7
rc206.100	1105.4	**5**	9.508	o	1051.1	7
rc207.100	1001.1	**5**	10.699	o	962.9	6
rc208.100	**790.4**	4	29.252	h	828.14	3

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions, the third last column indicates if the best known solution is optimal (o) or heuristic (h).

As the results in table 5.13 show, the algorithm only improves the best known data for the instances with less than 800 customers. This could be because four of the six 800 categories was unable to be divided, and the division yields some improvement on the results. The most improvements have occurred in the 200 customer instances, which also indicates that the maximum number of customers per division should not be to large, and the larger instances cannot yield this using only 10 regions.

All the better results use a lot more vehicles than the previously reported best results up to 3 times as many vehicles. In a real life setting a solution which was less than 10 % better and used 3 times as many vehicles would probably not be considered a good solution. Because I have chosen to optimize the total number of driven kilometers, I report these results.

Table 5.13: Results for Hombergers instances with improved cost.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C1_2_2.TXT	2879.3	21	14.51	2917,89	18
R1_2_1.TXT	4965.7	24	16.023	5024,65	19
R2_2_1.TXT	3619.2	14	15.813	4483,16	4
R2_2_2.TXT	3287.6	13	17.217	3621,20	4
RC2_2_1.TXT	2980.1	13	15.975	3099,53	6
RC2_2_2.TXT	2714.1	11	17.133	2825,24	5
RC2_2_3.TXT	2553.9	7	17.972	2603,83	4
RC2_2_5.TXT	2685.5	10	16.208	2911,46	4
RC2_2_6.TXT	2684.3	8	15.833	2975,13	4
R2_4_1.TXT	8150.9	27	34.14	9213,68	8
RC2_4_2.TXT	6311.5	20	34.867	6355,59	9
R2_6_1.TXT	16796.7	36	53.208	18291,18	11

Real world Data
The data form the Danish route planning company are tested on 4 different setups. Two versions of distance between the customers is used, the actual distance and the Euclidean distance. The time used to traverse the edge between two customers is in one half of the test calculated as the distance travelled at 65 km/h, and in the other half the actual time used to traverse the edges are used. The distribution of the data is shown in figure 5.1, with the five different days shown in different colors.

For Day 1 in the real world data it was not possible to do the division using the Euclidean distance, thus the comparison for Day 1 is with the undivided instance when it is reported Euclidean. The results for the real world data is shown in table 5.14.

The routes for the four different settings for day 2 of the real life data is in appendix D the rest of the routes are put on the CD-rom. The routes created for day two, with the actual distance and time consumption are shown in figure 5.2. The edges combining the depot and the routes are left out, because they are irrelevant for the results. Figure 5.2 shows that there is only one route going to Langeland, and there is also only on route traveling to Djursland.

The routes created when using the Euclidean distance is shown in figure 5.3.

Figure 5.1 Distribution of the customers in the real world data

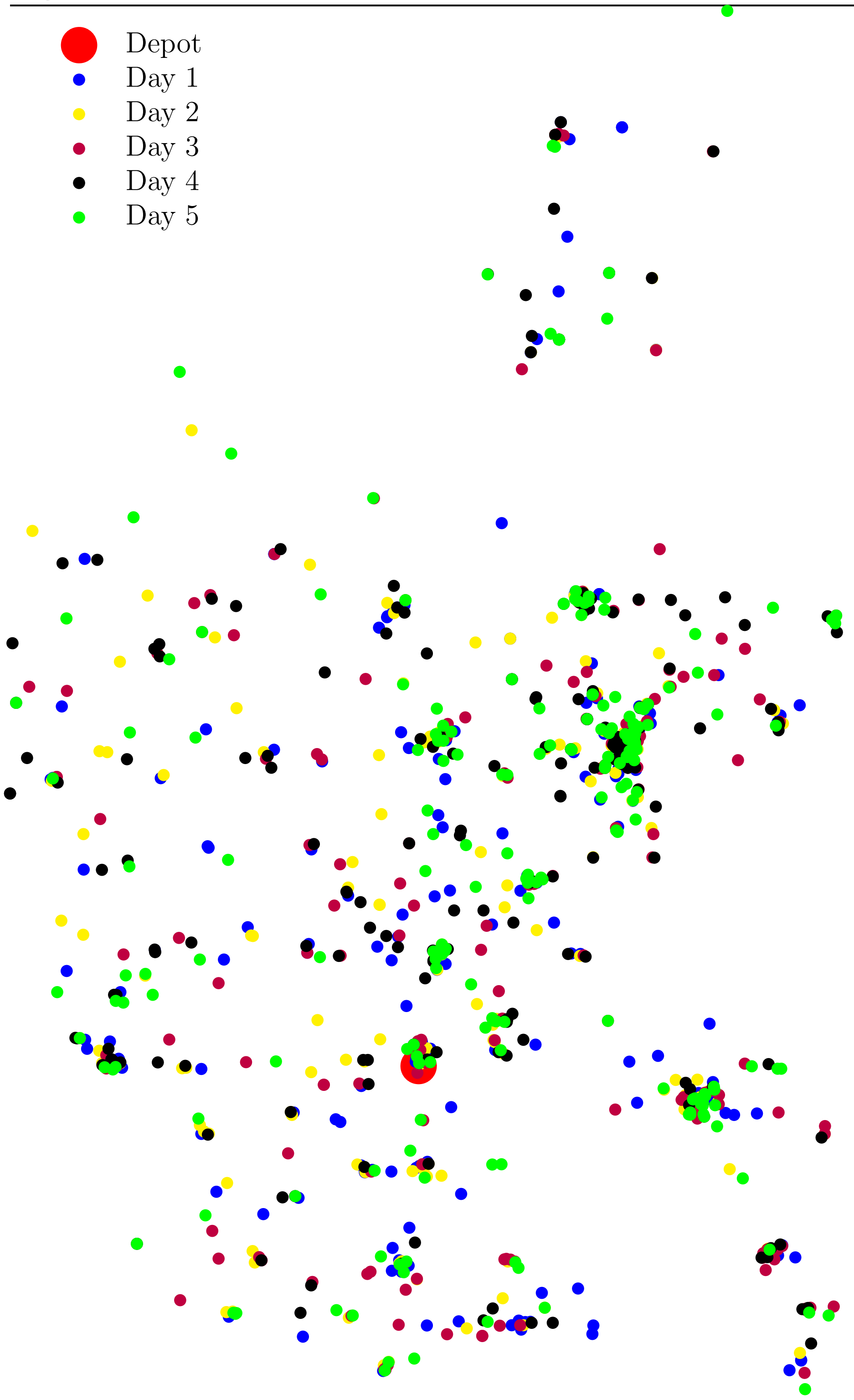


Figure 5.2 Solution for Day 2 in the real world data with actual distance

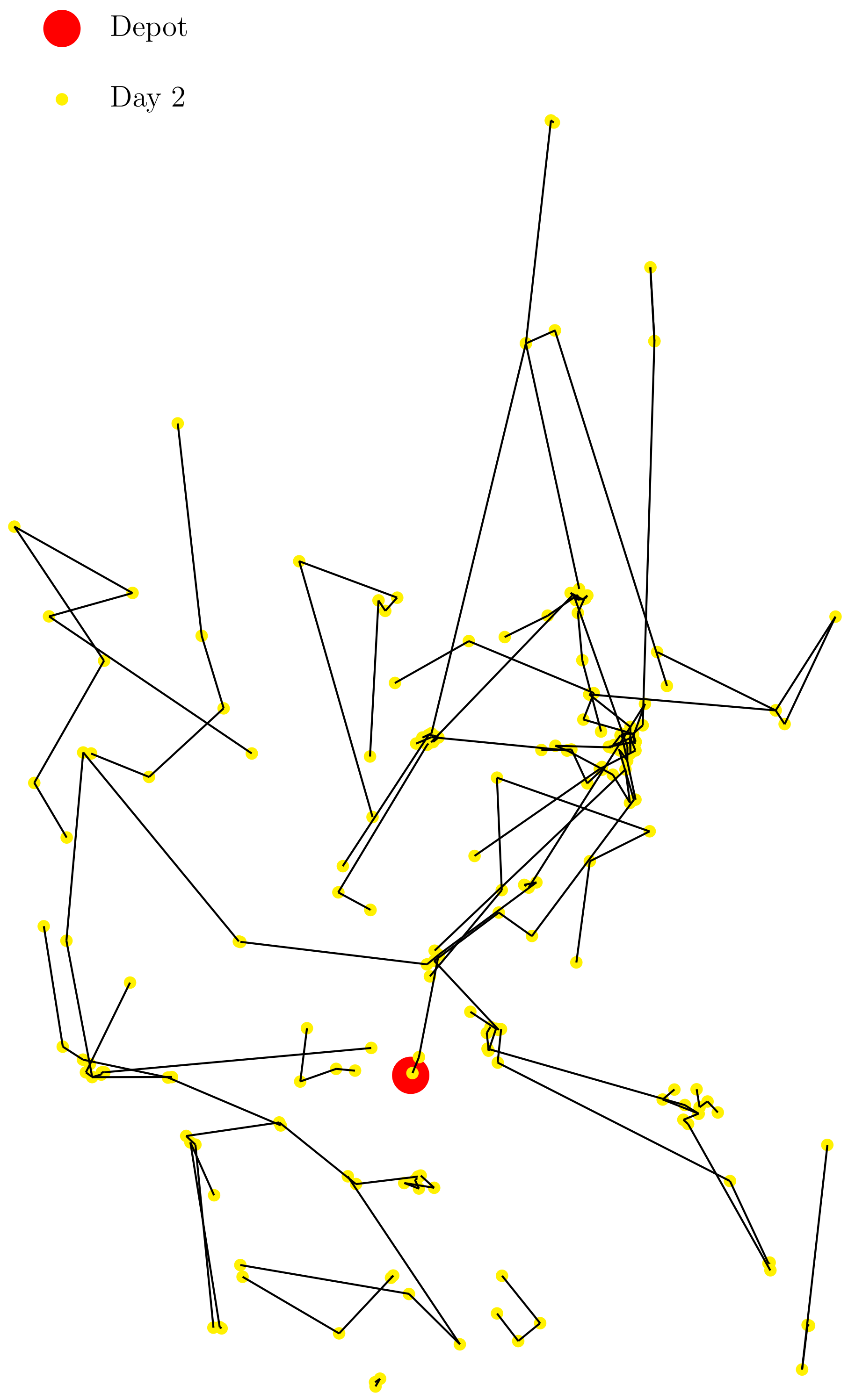


Figure 5.3 Solution for Day 2 in the real world data with Euclidean distance

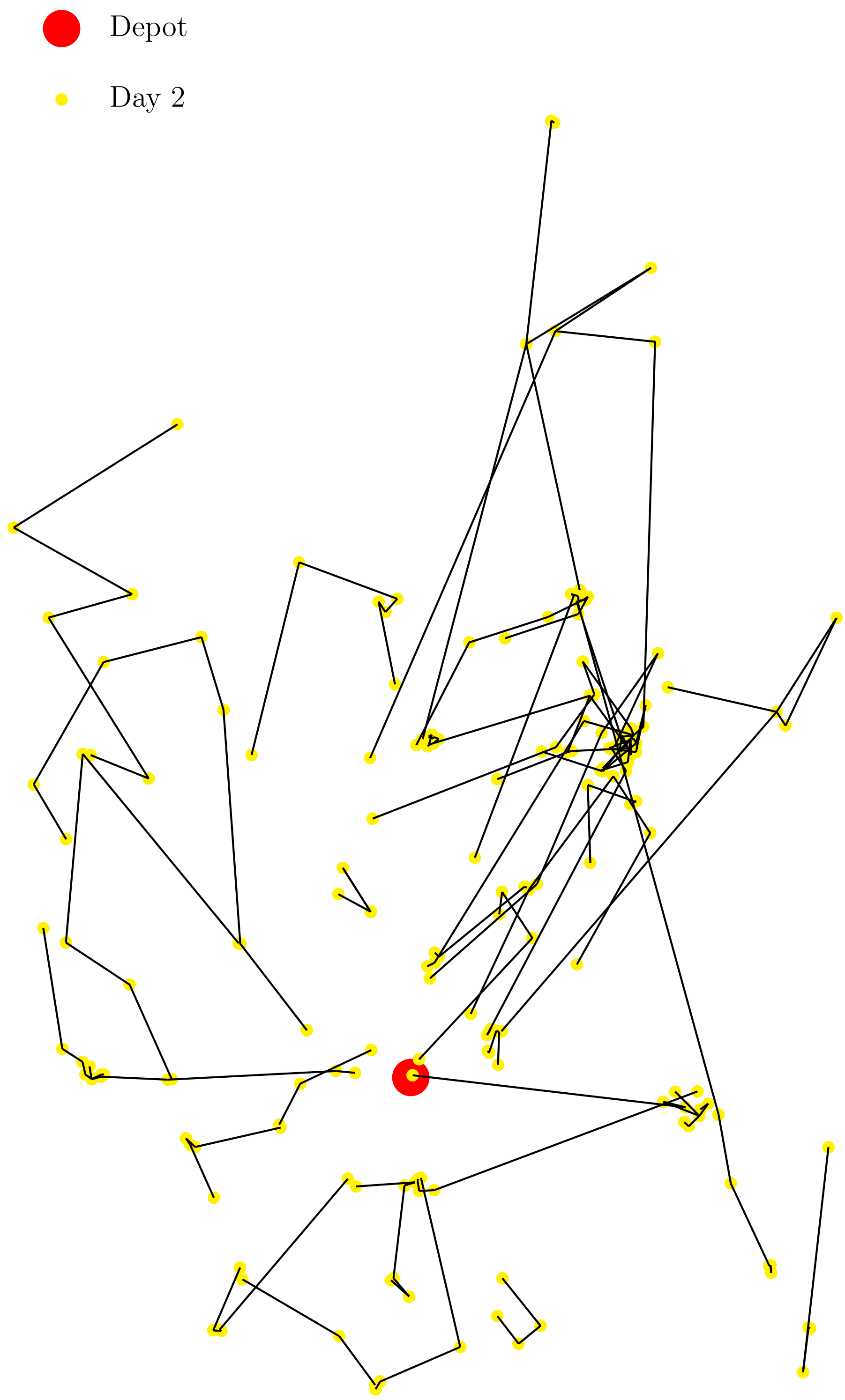


Table 5.14: Real World Data

	Real Distance			Euclidian distance		
	Solution	Vehicles	time (s)	Solution	Vehicles	time (s)
	Day 1					
Actual time	11546.9	47	19.65	12052.7	47	19.89
65 km/h	11259.3	45	18.01	11331.4	45	17.93
	Day 2					
Actual time	10665.3	41	16.69	8022.1	40	16.39
65 km/h	10538.2	39	15.5	8261.8	39	14.88
	Day 3					
Actual time	9894	38	15.76	7977.2	36	15.54
65 km/h	9834.9	35	14.42	7802.9	35	13.88
	Day 4					
Actual time	11976.4	43	17.64	9418.3	42	17.63
65 km/h	11715.2	41	16.46	9398.8	41	16.15
	Day 5					
Actual time	11210.2	38	16.38	8282.8	39	14.85
65 km/h	10656.4	38	15.33	8682.8	38	16.19

From figure 5.3 it is seen that some of the routes cross Little Belt to get from Odense to Århus, and one route will go from Vejle to Djursland, this implies that the use of the actual distance do a difference.
The actual route cost for day 2 with Euclidean distance is 10573.1 km which is a bit better than the cost of the test with the actual distance. The actual distance of the routes created using the Euclidean distance is 30% higher than the one reported by the original test.

It can be difficult to find good routes for a region like Denmark, but with the division strategy used in this thesis acceptable routes are created.

Chapter 6

Concluding remarks

In this thesis I have developed an ALNS framework for solving the VRPTW. The initial solution to the VRPTW was made by dividing the plane into regions using Voronoi Diagrams. The framework had better results when the test instance had clustered customers, than when the customers were randomly distributed across the plane. I found the optimal solution for the simple test instances, including one instance where the initial solution was the optimal. The algorithm did not work as well as hoped on the larger instances by Homberger [5]. This is probably because the initial solution did not partition the problem in small enough subproblems.

The ALNS algorithm get the best results when the starting solution is better. This implies that work should be put into improving the start solutions for the ALNS, by either subdividing more or optimizing before the solution is handed to the ALNS algorithm.

With this algorithm better results was reported for the heuristic solution to four of the five instances by Solomon [4] previously not solved to optimality. I also succeeded in finding better solutions to 12 of the instances by Homberger [5].

The real world data shows that the routes created sends one vehicle to each of the small islands, which was the hope. The largest time usage reported was 105 seconds which is well below the 5 minutes which was aimed for.

The use of regions proved to be more important than routes for the removal heuristic proposed by Shaw [18]. In all the use of regions can be described as a success.

6.1 Future work

The original idea was to solve the problem in the regions, and then combine these solutions. The combination of the routes from the different regions were not combined before the solution was given to the ALNS, neither was the local solutions sought optimized, thus I will propose that this line of work should be looked into. The algorithm developed in this thesis holds information of which regions are neighbors, this information could be used to find routes which are good candidates for combination.

The ALNS could be run in each region, and when the solutions in each region was good enough or had run for a while, the solutions could be combined, and this solution could be used as the initial solution for a ALNS on the full problem. For the large instances the subdivision could be subdivided once more, in order to get small enough solution spaces to search more thoroughly around these. Because the solution for the largest instances used less than two minutes of computational time, and each of the smallest instances used less than 15 seconds of computational time, it will be reasonable to assume that a two stage ALNS will be able to find the results for the largest instances in less than 15 minutes.

In this thesis the number of regions have been limited to 10, but if the datastructures used to divide the problem was improved, it is possible that an increase in the number of regions will yield even better results. The creation of the initial solution for the 100 customer problems was less than 1 second. If the initial solution was improved a lot by slowing things down the resulting algorithm would most likely improve.

It would be interesting to see if the knowledge of the infrastructure will do any difference, I have shown that the actual distance gives different results than those using the Euclidean distance. If the infrastructural hubs can be found the Voronoi Diagram created using these as centers could result in even better solutions.

List of Figures

List of Algorithms

Bibliography

[1] G. B. Dantzig and R. H. Ramser. The truck dispatching problem. *Management Science 6, pp. 80-91*, 1959.

[2] H. G. M. Pullen and M. H. J. Webb. A computer application to a transport scheduling problem. *The Computer Journal*, 10(1):10–13, May 1967. ISSN 0010-4620.

[3] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2001.

[4] M. M. Solomon. Vrptw benchmark problems. `http://w.cba.neu.edu/~msolomon/problems.htm`, 1983.

[5] J. Homberger. Extended solomon's vrptw instances. `http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm`, 1999.

[6] J. F. Bard, G. Kontoravdis, and G. Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportion Science 36, pp. 250-269*, 2002.

[7] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks 14, pp. 545-565*, 1984.

[8] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research 40, pp. 342-354*, 1992.

[9] N. Kohl. *Exact methods for time constrained routing and related scheduling problems*. PhD thesis, DTU IMM Lyngby, Denmark, 1995.

[10] J. Larsen. *Parallelization of the Vehicle Routing Problem with Time Windows*. PhD thesis, DTU IMM Lyngby, Denmark, 1999.

[11] W. Cook and J. L. Rich. A parallel cutting plane algorithm for the vehicle routing problem with time windows. Technical report, Computational and Applied Mathematics, Rice University, Houston, TX, 1999.

[12] S. Irnich and D. Villeneuve. The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. Technical report, Les Cahiers du GERAD, HEC Montral, Quebec, Canada, 2003.

[13] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle routing problem with time windows. *Operations Research*, 56(2):497–511, 2008. doi: 10.1287/opre.1070.0449.

[14] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2):254–265, 1987. ISSN 0030-364X.

[15] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, May 1993. URL `http://ideas.repec.org/a/eee/ejores/v66y1993i3p331-340.html`.

[16] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Handbook of metaheuristics*, chapter 16. Hyper-heuristics: an emerging direction in modern search technology. Springer, 2003.

[17] S. Røpke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472, 2006.

[18] P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems, 1997.

[19] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. *CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming)*, 1520:417–431, 1998.

[20] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry.* Springer, February 2000. ISBN 3540656200.

[21] Y. Kwon, J. Kim, J. Seo, D. Lee, and D. Kim. A tabu search algorithm using the voronoi diagram for the capacitated vehicle routing problem. *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on, pp 480-488*, 2007.

[22] J.E. Beasley and N. Christofides. Vehicle routing with a sparse feasibility graph. *European Journal of Operational Research 98, pp.499-511*, 1997.

[23] G. Ioannou, M. Kritikos, and G. Prastacos. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *The Journal of the Operational Research Society*, 52(5):523–537, 2001. ISSN 01605682. URL `http://www.jstor.org/stable/253988`.

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[25] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985.

[26] Fred Glover. Tabu Search–Part I. *INFORMS JOURNAL ON COMPUTING*, 1(3):190–206, 1989. URL `http://joc.journal.informs.org/cgi/content/abstract/1/3/190`.

[27] Fred Glover. Tabu Search–Part II. *INFORMS JOURNAL ON COMPUTING*, 2(1):4–32, 1990. URL `http://joc.journal.informs.org/cgi/content/abstract/2/1/4`.

[28] B. E. Gillett and L. R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974. ISSN 0030364X. URL `http://www.jstor.org/stable/169591`.

[29] S. O'Sullivan. c++ version of steven fortunes voronoi diagram implementation. `http://mapmanager.cvs.sourceforge.net/mapmanager/src/voronoi/`, 2003.

[30] S. Fortune. 2-dimensional voronoi diagram, delaunay triangulation, sweepline implementation. `http://netlib.sandia.gov/voronoi/index.html`, 1994.

[31] Sintef. Results for hombergers vrptw benchmark instances, 2001-2004. URL `http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html`.

Appendix A

Division vs. full problem

In this Appendix are the results of the initial solution for the 8 insertion heuristics described in chapter 2. Each of the heuristics are run twice on each instance, once on the entire problem, and once on the divided problem. The division used is the Voronoi Diagrams, where the centers have been found by Kruskal's algorithm.

Table A.1: Data for the simple greedy insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	10	8768	10	8273
c105.sol	10	9372	11	8800
c201.sol	44	29738	11	11164
c205.sol	29	22405	11	12574
C1_2_1.TXT	21	33989	23	33496
C1_2_8.TXT	21	37653	23	35512
C2_2_1.TXT	36	53696	17	33716
C2_2_8.TXT	9	26343	18	37802
r101.sol	22	21454	29	23603
r105.sol	19	20101	20	19071
r201.sol	11	14739	11	16165
r205.sol	9	15980	11	15454
rc101.sol	18	21897	24	26340
rc105.sol	19	22559	22	24216
rc201.sol	17	19003	11	18221
rc205.sol	12	16289	11	18147

Table A.2: Data for the memory using greedy insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	10	8768	10	8273
c105.sol	10	9372	11	8800
c201.sol	43	28952	11	11164
c205.sol	29	22405	11	12574
C1_2_1.TXT	21	33989	23	33496
C1_2_8.TXT	21	37653	23	35512
C2_2_1.TXT	35	52525	17	33716
C2_2_8.TXT	9	26343	18	37802
r101.sol	22	21454	29	23603
r105.sol	18	19308	20	19071
r201.sol	11	14739	11	16165
r205.sol	9	15645	11	15454
rc101.sol	18	21897	24	26340
rc105.sol	19	22559	22	24216
rc201.sol	17	19003	11	18221
rc205.sol	12	16289	11	18147

Table A.3: Data for the regret insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	17	16715	12	9978
c105.sol	27	24781	12	9978
c201.sol	5	11260	12	12259
c205.sol	5	11803	13	13355
C1_2_1.TXT	34	60657	30	45368
C1_2_8.TXT	28	64909	28	51269
C2_2_1.TXT	18	41908	16	36226
C2_2_8.TXT	10	47588	17	38855
r101.sol	23	21670	29	23327
r105.sol	21	22420	20	20609
r201.sol	7	18276	13	16785
r205.sol	5	19725	11	15255
rc101.sol	24	27712	23	25506
rc105.sol	20	22037	22	21421
rc201.sol	7	23140	12	19273
rc205.sol	7	25010	11	18105

Table A.4: Data for the regret 3 insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	11	9668	12	8947
c105.sol	14	13246	12	9058
c201.sol	5	7508	13	12242
c205.sol	5	7308	13	12462
C1_2_1.TXT	25	44334	26	38840
C1_2_8.TXT	25	48312	25	38270
C2_2_1.TXT	15	24723	17	32128
C2_2_8.TXT	16	27437	17	35771
r101.sol	23	21977	28	21944
r105.sol	17	17817	21	18356
r201.sol	16	15142	12	16063
r205.sol	6	12419	15	14843
rc101.sol	19	23458	23	23149
rc105.sol	18	20850	22	21057
rc201.sol	14	16516	12	18079
rc205.sol	12	16501	11	17442

Table A.5: Data for the regret 4 insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	11	11042	12	8947
c105.sol	13	12714	12	9058
c201.sol	7	7917	13	12242
c205.sol	6	8890	13	12462
C1_2_1.TXT	25	44066	26	38816
C1_2_8.TXT	27	50667	26	39174
C2_2_1.TXT	10	22643	17	32194
C2_2_8.TXT	12	27931	18	36080
r101.sol	20	19366	28	21814
r105.sol	19	18884	21	18321
r201.sol	16	15729	12	16063
r205.sol	8	11792	15	14843
rc101.sol	18	21082	23	23149
rc105.sol	18	21275	21	20991
rc201.sol	26	23231	12	18079
rc205.sol	10	15881	11	17442

Table A.6: Data for the regret 5 insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	14	13270	12	8947
c105.sol	12	12035	12	9058
c201.sol	18	14549	13	12242
c205.sol	6	9754	13	12462
C1_2_1.TXT	27	44252	26	38816
C1_2_8.TXT	25	50812	26	39174
C2_2_1.TXT	33	46241	17	32194
C2_2_8.TXT	15	28669	18	36080
r101.sol	21	20228	28	21464
r105.sol	17	17426	21	18359
r201.sol	16	15309	12	16063
r205.sol	12	13249	15	14843
rc101.sol	18	20791	23	23149
rc105.sol	17	20225	21	20991
rc201.sol	25	22523	12	18079
rc205.sol	10	15725	11	17442

Table A.7: Data for the regret 6 insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	12	12938	12	8947
c105.sol	12	13024	12	9058
c201.sol	17	13765	13	12242
c205.sol	6	9754	13	12462
C1_2_1.TXT	26	46217	26	38816
C1_2_8.TXT	26	51971	26	39174
C2_2_1.TXT	17	30275	17	32194
C2_2_8.TXT	11	30563	18	36080
r101.sol	21	20721	28	21455
r105.sol	17	17429	21	18422
r201.sol	16	15455	12	16063
r205.sol	10	12780	15	14843
rc101.sol	18	22345	23	23149
rc105.sol	19	21366	21	20991
rc201.sol	26	23231	12	18079
rc205.sol	10	14833	11	17442

Table A.8: Data for the sequential greedy insertion algorithm

	No division		Kruskal Voronoi	
Instance	# Vehicles	distance	# Vehicles	distance
c101.sol	10	8768	10	8273
c105.sol	10	9531	11	8800
c201.sol	4	7905	11	11303
c205.sol	5	8763	11	12574
C1_2_1.TXT	23	42038	23	33496
C1_2_8.TXT	22	43814	24	36509
C2_2_1.TXT	8	24885	16	32287
C2_2_8.TXT	8	28116	14	35708
r101.sol	22	21640	29	23533
r105.sol	19	20585	19	19478
r201.sol	8	14967	11	16230
r205.sol	5	15602	10	15564
rc101.sol	20	24842	24	24757
rc105.sol	19	23563	25	24618
rc201.sol	10	17398	10	18845
rc205.sol	9	16549	10	18407

Appendix B

New best Routes

B.1 RC208

Route 0, cost: 1863, demand: 519: 0 → 69 → 98 → 88 → 53 → 9 → 11 → 15 → 16 → 17 → 47 → 14 → 12 → 78 → 73 → 79 → 7 → 6 → 2 → 8 → 46 → 4 → 45 → 5 → 3 → 1 → 100 → 70 → 68
Route 1, cost: 2351, demand: 475: 0 → 90 → 82 → 99 → 52 → 86 → 87 → 59 → 75 → 58 → 77 → 25 → 23 → 21 → 48 → 18 → 19 → 49 → 20 → 22 → 24 → 57 → 74 → 97 → 13 → 10 → 60 → 55
Route 2, cost: 2189, demand: 394: 0 → 65 → 83 → 64 → 95 → 67 → 62 → 50 → 34 → 31 → 29 → 27 → 26 → 28 → 30 → 32 → 33 → 76 → 89 → 63 → 85 → 51 → 84 → 56 → 91 → 66
Route 3, cost: 1501, demand: 336: 0 → 92 → 94 → 93 → 71 → 81 → 61 → 42 → 44 → 43 → 40 → 36 → 35 → 37 → 38 → 39 → 41 → 72 → 54 → 96 → 80
total cost =790.4, Total routes = 4
found in iteration 18743 after 22 seconds

Least Vehicles
Route 0, cost: 3928, demand: 790: 0 → 65 → 82 → 11 → 15 → 16 → 14 → 12 → 88 → 61 → 41 → 39 → 38 → 36 → 40 → 44 → 42 → 2 → 5 → 3 → 1 → 45 → 4 → 46 → 8 → 7 → 6 → 60 → 79 → 73 → 78 → 17 → 47 → 10 → 9 → 13 → 97 → 58 → 74 → 24 → 56 → 91 → 80

Route 1, cost: 3810, demand: 543: 0 → 92 → 95 → 51 → 76 → 64 → 81 → 72 → 71 → 67 → 62 → 50 → 34 → 31 → 29 → 27 → 26 → 28 → 30 → 32 → 33 → 89 → 63 → 85 → 84 → 94 → 93 → 96 → 54 → 37 → 35 → 43 → 70 → 100 → 55 → 68
Route 2, cost: 1749, demand: 391: 0 → 69 → 98 → 53 → 99 → 52 → 86 → 87 → 59 → 75 → 77 → 25 → 23 → 21 → 48 → 18 → 19 → 49 → 20 → 22 → 57 → 83 → 66 → 90
total cost =948.7, Total routes = 3
found in iteration 9054 after 11 seconds
rc208.sol&790.4&4&29.252

B.2 R204

Least Vehicles Route 0, cost: 3530, demand: 631: 0 → 2 → 57 → 41 → 22 → 75 → 56 → 23 → 67 → 39 → 54 → 76 → 50 → 51 → 9 → 81 → 33 → 79 → 78 → 34 → 35 → 71 → 65 → 66 → 20 → 30 → 32 → 90 → 63 → 64 → 49 → 36 → 47 → 46 → 45 → 17 → 61 → 91 → 100 → 98 → 37 → 95 → 94 → 6 → 89
Route 1, cost: 2486, demand: 483: 0 → 27 → 69 → 1 → 70 → 10 → 31 → 88 → 62 → 11 → 19 → 48 → 7 → 82 → 8 → 83 → 84 → 5 → 53 → 40 → 21 → 73 → 74 → 72 → 4 → 55 → 25 → 24 → 29 → 3 → 77 → 68 → 80 → 12 → 26 → 28
Route 2, cost: 1580, demand: 344: 0 → 52 → 18 → 60 → 96 → 92 → 42 → 15 → 43 → 14 → 38 → 44 → 86 → 16 → 85 → 93 → 99 → 59 → 97 → 87 → 13 → 58
total cost =759.6, Total routes = 3
found in iteration 20234 after 22 seconds

Route 0, cost: 3441, demand: 660: 0 → 27 → 52 → 7 → 82 → 48 → 19 → 11 → 62 → 88 → 31 → 69 → 1 → 50 → 76 → 3 → 79 → 33 → 51 → 9 → 81 → 78 → 34 → 35 → 71 → 65 → 66 → 20 → 30 → 70 → 10 → 32 → 90 → 63 → 64 → 49 → 36 → 47 → 46 → 45 → 17 → 61 → 91 → 100 → 37 → 98 → 59 → 96 → 6
Route 1, cost: 2714, demand: 536: 0 → 94 → 95 → 92 → 97 → 42 → 15 → 43 → 14 → 44 → 38 → 86 → 16 → 85 → 93 → 99 → 5 → 84 → 8 → 83 → 60 → 18 → 89 → 40 → 21 → 73 → 74 → 72 → 4 → 55 → 25 → 24 → 29 → 77 → 68 → 80 → 12 → 28
Route 2, cost: 1278, demand: 248: 0 → 26 → 54 → 39 → 67 → 23 → 56 → 75 → 22 → 41 → 57 → 2 → 87 → 13 → 58
Route 3, cost: 88, demand: 14 : 0 → 53
total cost =752.1, Total routes = 4
found in iteration 23084 after 25 seconds
r204.sol&752.1&4&27.76

B.3 R208

Route 0, cost: 3708, demand: 610: 0 → 26 → 56 → 39 → 67 → 23 → 75 → 22 → 41 → 2 → 57 → 15 → 43 → 42 → 87 → 97 → 96 → 99 → 5 → 84 → 17 → 45 → 8 → 46 → 47 → 36 → 49 → 64 → 63 → 90 → 32 → 20 → 66 → 65 → 71 → 35 → 34 → 78 → 29 → 24 → 80 → 68 → 77 → 50 → 1 → 28
Route 1, cost: 2029, demand: 461: 0 → 27 → 69 → 31 → 88 → 7 → 82 → 48 → 19 → 11 → 62 → 10 → 70 → 30 → 51 → 9 → 81 → 33 → 79 → 3 → 76 → 12 → 54 → 55 → 25 → 4 → 72 → 74 → 73 → 21 → 40 → 58
Route 2, cost: 1426, demand: 373 :0 → 52 → 18 → 83 → 60 → 89 → 6 → 94 → 95 → 92 → 59 → 93 → 85 → 61 → 16 → 86 → 38 → 14 → 44 → 91 → 100 → 98 → 37 → 13
Route 3, cost: 88, demand: 14 :0 → 53
total cost =725.1, Total routes = 4
found in iteration 17185 after 23 seconds

Least Vehicles Route 0, cost: 4164, demand: 898: 0 → 27 → 52 → 18 → 7 → 82 → 48 → 47 → 19 → 11 → 62 → 88 → 31 → 69 → 1 → 50 → 76 → 12 → 26 → 21 → 72 → 22 → 41 → 23 → 67 → 39 → 54 → 80 → 68 → 3 → 79 → 81 → 9 → 71 → 66 → 65 → 35 → 34 → 78 → 29 → 24 → 55 → 25 → 4 → 56 → 75 → 74 → 73 → 40 → 58 → 13 → 37 → 100 → 91 → 85 → 93 → 96 → 6 → 89
Route 1, cost: 3316, demand: 560: 0 → 94 → 95 → 92 → 98 → 61 → 16 → 86 → 38 → 44 → 14 → 42 → 43 → 15 → 57 → 2 → 53 → 87 → 97 → 59 → 99 → 5 → 60 → 83 → 84 → 17 → 45 → 8 → 46 → 36 → 49 → 64 → 63 → 90 → 32 → 10 → 70 → 30 → 20 → 51 → 33 → 77 → 28
total cost =748, Total routes = 2
found in iteration 23835 after 34 seconds
../../../data/solomon/r208.sol&725.1&4&31.886

B.4 R211

Route 0, cost: 2715, demand: 590: 0 → 53 → 40 → 94 → 95 → 98 → 85 → 61 → 16 → 86 → 38 → 44 → 14 → 43 → 15 → 22 → 74 → 56 → 39 → 67 → 25 → 55 → 4 → 54 → 80 → 68 → 77 → 26 → 58 → 13 → 97 → 37 → 100 → 91 → 93 → 96 → 89
Route 1, cost: 2709, demand: 417: 0 → 27 → 69 → 31 → 88 → 62 → 11 → 64 → 63 → 90 → 30 → 51 → 9 → 81 → 33 → 79 → 3 → 50 → 76 → 24 → 29 → 78 → 34 → 35 → 71 → 65 → 66 → 20 → 32 → 10 → 70 → 1
Route 2, cost: 2605, demand: 451: 0 → 28 → 12 → 21 → 73 → 72 → 75 → 23 → 41 → 2 → 57 → 42 → 87 → 92 → 59 → 99 → 5 → 83 → 18 → 52 → 7 → 82 → 48 → 19 → 49 → 36 → 47 → 46 → 8 → 45 → 17 → 84 → 60 → 6
total cost =802.9, Total routes = 3
found in iteration 18864 after 25 seconds
r211.sol&802.9&3&33.881

Route 0, cost: 2188, demand: 414: 0 → 53 → 40 → 21 → 73 → 72 → 75 → 23 → 67 → 39 → 25 → 55 → 54 → 24 → 29 → 80 → 68 → 77 → 3 → 79 → 78 → 34 → 35 → 66 → 20 → 32 → 10 → 70 → 1
Route 1, cost: 2047, demand: 460: 0 → 27 → 69 → 31 → 30 → 51 → 71 → 65 → 9 → 81 → 33 → 50 → 76 → 12 → 28 → 2 → 57 → 87 → 97 → 92 → 37 → 100 → 91 → 85 → 93 → 99 → 96 → 94 → 13 → 58
Route 2, cost: 1576, demand: 275: 0 → 6 → 95 → 59 → 98 → 61 → 16 → 86 → 38 → 44 → 14 → 42 → 43 → 15 → 41 → 22 → 74 → 56 → 4 → 26
Route 3, cost: 2018, demand: 309: 0 → 52 → 18 → 83 → 8 → 82 → 7 → 88 → 62 → 19 → 11 → 90 → 63 → 64 → 49 → 36 → 47 → 48 → 46 → 45 → 17 → 84 → 5 → 60 → 89
total cost =782.9, Total routes = 4
found in iteration 20994 after 26 seconds
r211.sol&782.9&4&31.028

B.5 C1_2_2

Route 0, cost: 1843, demand: 180: 0 → 57 → 180 → 84 → 191 → 125 → 4 → 82 → 72 → 60 → 36
Route 1, cost: 1031, demand: 120: 0 → 48 → 26 → 14 → 96 → 130 → 28 → 68 → 76
Route 2, cost: 605, demand: 170: 0 → 113 → 155 → 78 → 175 → 13 → 43 → 2 → 90 → 67 → 39 → 107
Route 3, cost: 980, demand: 160: 0 → 101 → 144 → 119 → 166 → 35 → 126 → 71 → 9 → 1 → 99
Route 4, cost: 1498, demand: 180: 0 → 178 → 50 → 156 → 112 → 168 → 79 → 29 → 87 → 42 → 123 → 133
Route 5, cost: 2016, demand: 200: 0 → 177 → 3 → 88 → 8 → 186 → 127 → 98 → 157 → 135 → 183
Route 6, cost: 1310, demand: 170: 0 → 118 → 83 → 143 → 176 → 33 → 121 → 165 → 188 → 108 → 17
Route 7, cost: 1437, demand: 180: 0 → 190 → 5 → 10 → 193 → 46 → 128 → 106 → 167 → 34 → 95 → 158
Route 8, cost: 926, demand: 110: 0 → 116 → 12 → 129 → 11 → 6 → 122 → 139
Route 9, cost: 934, demand: 200: 0 → 20 → 41 → 85 → 80 → 31 → 25 → 172 → 77 → 110 → 162
Route 10, cost: 1643, demand: 180: 0 → 30 → 120 → 19 → 192 → 196 → 97 → 195 → 92 → 59 → 149
Route 11, cost: 1797, demand: 200: 0 → 114 → 159 → 38 → 150 → 22 → 151 → 16 → 140 → 187 → 142 → 111 → 63 → 56
Route 12, cost: 1421, demand: 140: 0 → 93 → 55 → 58 → 184 → 199 → 37 → 81 → 138 → 137
Route 13, cost: 2077, demand: 190: 0 → 32 → 171 → 65 → 86 → 115 → 94 → 51 → 174 → 136 → 189 → 124
Route 14, cost: 1482, demand: 150: 0 → 170 → 134 → 152 → 40 → 153 → 169 → 89 → 105 → 198
Route 15, cost: 657, demand: 170: 0 → 45 → 27 → 173 → 154 → 24 → 61 → 100 → 64 → 179 → 109
Route 16, cost: 1766, demand: 200: 0 → 21 → 23 → 182 → 75 → 163 → 194 → 145 → 52 → 15 → 74
Route 17, cost: 1967, demand: 190: 0 → 161 → 104 → 18 → 54 → 185 → 132 → 7 → 181 → 117 → 49 → 53
Route 18, cost: 1181, demand: 200: 0 → 164 → 66 → 147 → 160 → 47 → 91 → 70 → 73

Route 19, cost: 634, demand: 110: 0 → 62 → 131 → 44 → 146 → 102
Route 20, cost: 1588, demand: 130: 0 → 148 → 103 → 197 → 141 → 69 →
200
total cost =2879.3, Total routes = 21
found in iteration 18786 after 11 seconds
../../../data/Nsolomon/C1_2_2.TXT&2879.3&21&14.51

B.6 R1_2_1

Route 0, cost: 2133, demand: 151:0 → 157 → 93 → 199 → 135 → 80 → 64 → 109 → 182 → 91
Route 1, cost: 1916, demand: 194:0 → 168 → 123 → 113 → 31 → 185 → 169 → 111 → 105
Route 2, cost: 1972, demand: 175:0 → 63 → 44 → 36 → 3 → 5 → 133 → 195 → 180 → 28
Route 3, cost: 1836, demand: 181:0 → 140 → 57 → 19 → 95 → 178 → 23 → 22 → 47 → 82 → 74
Route 4, cost: 3015, demand: 138:0 → 145 → 35 → 152 → 34 → 98 → 136 → 142 → 159 → 102 → 10 → 198
Route 5, cost: 2368, demand: 130:0 → 155 → 67 → 150 → 137 → 68 → 25 → 79 → 156 → 78 → 119
Route 6, cost: 1866, demand: 155:0 → 46 → 54 → 89 → 125 → 114 → 39 → 37 → 164
Route 7, cost: 1487, demand: 191:0 → 196 → 38 → 62 → 194 → 118 → 88 → 183 → 52 → 24
Route 8, cost: 1388, demand: 131:0 → 73 → 60 → 99 → 121 → 32 → 138 → 83 → 45
Route 9, cost: 2390, demand: 178:0 → 4 → 15 → 184 → 153 → 193 → 151 → 129 → 141 → 191 → 110
Route 10, cost: 2162, demand: 74:0 → 188 → 71 → 144 → 186 → 179 → 190
Route 11, cost: 2097, demand: 189:0 → 149 → 81 → 161 → 92 → 48 → 139 → 187 → 170 → 42
Route 12, cost: 2417, demand: 153:0 → 87 → 115 → 16 → 76 → 17 → 40 → 128 → 59
Route 13, cost: 2329, demand: 107:0 → 106 → 85 → 94 → 12 → 58 → 65 → 43
Route 14, cost: 2721, demand: 155:0 → 55 → 50 → 66 → 160 → 116 → 49 → 27 → 61 → 75
Route 15, cost: 2096, demand: 108:0 → 117 → 86 → 131 → 130 → 11 → 172
Route 16, cost: 1574, demand: 143:0 → 132 → 18 → 14 → 146 → 124 → 134 → 103 → 154
Route 17, cost: 1816, demand: 95:0 → 72 → 147 → 166 → 33 → 167 → 70 → 107
Route 18, cost: 2346, demand: 151:0 → 30 → 20 → 163 → 100 → 84 → 90 → 29 → 96 → 174 → 97

Route 19, cost: 1155, demand: 86:0 → 7 → 41 → 175 → 56 → 9
Route 20, cost: 2060, demand: 149:0 → 158 → 51 → 143 → 176 → 165 → 69 → 181 → 101
Route 21, cost: 2231, demand: 157:0 → 13 → 173 → 104 → 6 → 189 → 177 → 127 → 77 → 108
Route 22, cost: 2528, demand: 171:0 → 112 → 2 → 8 → 162 → 171 → 122 → 53 → 192 → 200
Route 23, cost: 1754, demand: 151:0 → 1 → 120 → 26 → 21 → 126 → 148 → 197
total cost =4965.7, Total routes = 24
found in iteration 24550 after 16 seconds
../../../data/Nsolomon/R1_2_1.TXT&4965.7&24&16.023

B.7 R2_2_1

Route 0, cost: 2857, demand: 200:0 → 4 → 100 → 135 → 170 → 14 → 180 → 176 → 98 → 189 → 83 → 121 → 153 → 25 → 148
Route 1, cost: 3172, demand: 304:0 → 151 → 21 → 6 → 40 → 32 → 44 → 146 → 193 → 27 → 159 → 187 → 7 → 156 → 22 → 55
Route 2, cost: 3114, demand: 239:0 → 41 → 47 → 65 → 139 → 88 → 31 → 132 → 195 → 15 → 29 → 147 → 160 → 155 → 1 → 140 → 115
Route 3, cost: 1799, demand: 201:0 → 23 → 77 → 152 → 118 → 19 → 43 → 103 → 9 → 129 → 125 → 42
Route 4, cost: 1487, demand: 187:0 → 34 → 63 → 104 → 165 → 3 → 144 → 158 → 111
Route 5, cost: 2830, demand: 256:0 → 143 → 119 → 71 → 48 → 197 → 167 → 178 → 191 → 185 → 101 → 127 → 112 → 80
Route 6, cost: 3281, demand: 265:0 → 45 → 8 → 26 → 161 → 11 → 141 → 66 → 142 → 105 → 123 → 130 → 198 → 93 → 64 → 10 → 126 → 169 → 53
Route 7, cost: 2755, demand: 186:0 → 131 → 62 → 179 → 137 → 117 → 162 → 94 → 149 → 108 → 76 → 188 → 182 → 54 → 190
Route 8, cost: 2637, demand: 251:0 → 183 → 181 → 78 → 150 → 57 → 90 → 68 → 13 → 114 → 24 → 136 → 50 → 154 → 171
Route 9, cost: 2447, demand: 172:0 → 74 → 192 → 36 → 33 → 35 → 177 → 134 → 28 → 97 → 52 → 39
Route 10, cost: 1748, demand: 266:0 → 164 → 73 → 79 → 91 → 133 → 17 → 16 → 89 → 58 → 173 → 5 → 86 → 99
Route 11, cost: 1569, demand: 292:0 → 18 → 67 → 60 → 184 → 200 → 145 → 102 → 196 → 122 → 107 → 138 → 96
Route 12, cost: 4336, demand: 373:0 → 168 → 95 → 194 → 128 → 38 → 124 → 166 → 157 → 84 → 174 → 70 → 113 → 82 → 106 → 2 → 110 → 12 → 109 → 30 → 20 → 46 → 85 → 87 → 163
Route 13, cost: 1055, demand: 89:0 → 61 → 92 → 75 → 69 → 186
Route 14, cost: 2282, demand: 232:0 → 56 → 116 → 199 → 120 → 51 → 172 → 81 → 59 → 37 → 49 → 175 → 72
total cost =3736.9, Total routes = 15
found in iteration 19159 after 13 seconds
../../../data/Nsolomon/R2_2_1.TXT&3736.9&15&15.937

B.8 R2_2_2

Route 0, cost: 3827, demand: 426:0 → 115 → 92 → 61 → 119 → 71 → 48 → 197 → 167 → 194 → 128 → 151 → 150 → 90 → 50 → 141 → 84 → 157 → 166 → 58 → 126 → 98 → 73 → 169 → 53
Route 1, cost: 2046, demand: 236:0 → 34 → 63 → 165 → 3 → 82 → 106 → 104 → 2 → 110 → 182 → 54
Route 2, cost: 3220, demand: 326:0 → 23 → 77 → 152 → 125 → 143 → 31 → 132 → 195 → 15 → 29 → 9 → 37 → 140 → 147 → 160 → 155 → 59 → 1 → 51 → 72 → 39
Route 3, cost: 2418, demand: 246:0 → 18 → 75 → 184 → 191 → 178 → 101 → 185 → 42 → 112 → 127 → 67 → 80
Route 4, cost: 2861, demand: 230:0 → 4 → 100 → 135 → 170 → 83 → 14 → 176 → 123 → 130 → 198 → 93 → 10 → 105 → 180 → 153 → 148
Route 5, cost: 1855, demand: 203:0 → 70 → 25 → 113 → 87 → 12 → 109 → 30 → 20 → 189 → 121 → 46 → 85 → 163
Route 6, cost: 2888, demand: 363:0 → 41 → 116 → 199 → 56 → 47 → 65 → 139 → 88 → 129 → 118 → 103 → 43 → 19 → 120 → 172 → 81 → 49 → 175 → 52
Route 7, cost: 1986, demand: 302:0 → 99 → 181 → 8 → 26 → 161 → 11 → 64 → 91 → 142 → 66 → 79 → 133 → 17 → 16 → 89 → 174 → 154
Route 8, cost: 3044, demand: 409:0 → 38 → 95 → 6 → 40 → 32 → 44 → 146 → 200 → 145 → 102 → 60 → 196 → 186 → 69 → 122 → 107 → 138 → 55 → 96
Route 9, cost: 1584, demand: 140:0 → 131 → 62 → 179 → 137 → 117 → 162 → 158 → 144 → 111
Route 10, cost: 1792, demand: 213:0 → 164 → 171 → 78 → 124 → 57 → 5 → 173 → 136 → 24 → 86 → 183
Route 11, cost: 2399, demand: 222:0 → 168 → 21 → 68 → 114 → 13 → 159 → 193 → 27 → 187 → 7 → 156 → 45 → 22
Route 12, cost: 2956, demand: 197:0 → 74 → 192 → 36 → 33 → 35 → 177 → 94 → 149 → 134 → 28 → 108 → 76 → 188 → 97 → 190
total cost =3287.6, Total routes = 13
found in iteration 19282 after 14 seconds
../../../data/Nsolomon/R2_2_2.TXT&3287.6&13&17.217

B.9 RC2_2_1

Route 0, cost: 3067, demand: 475:0 → 48 → 131 → 152 → 167 → 15 → 179 → 7 → 68 → 18 → 124 → 43 → 180 → 3 → 136 → 89 → 139 → 188 → 29 → 21 → 166 → 194 → 55 → 81 → 107 → 66 → 82
Route 1, cost: 2972, demand: 369:0 → 142 → 24 → 156 → 102 → 196 → 183 → 184 → 190 → 27 → 51 → 111 → 198 → 91 → 145 → 80 → 116 → 83 → 148 → 36
Route 2, cost: 2714, demand: 299:0 → 163 → 6 → 162 → 63 → 149 → 181 → 35 → 164 → 104 → 25 → 170 → 10 → 77 → 122 → 141 → 185 → 120 → 175
Route 3, cost: 2217, demand: 320:0 → 28 → 49 → 14 → 132 → 78 → 171 → 130 → 187 → 150 → 79 → 96 → 135 → 2 → 76 → 34 → 97 → 30 → 67
Route 4, cost: 2417, demand: 271:0 → 42 → 172 → 197 → 5 → 62 → 26 → 159 → 20 → 46 → 138 → 186 → 73 → 93 → 193 → 19 → 146 → 39
Route 5, cost: 2080, demand: 390:0 → 100 → 37 → 9 → 12 → 59 → 143 → 173 → 56 → 108 → 38 → 32 → 23 → 117 → 114 → 86 → 74 → 52 → 129 → 41 → 158 → 154
Route 6, cost: 2748, demand: 262:0 → 169 → 57 → 133 → 147 → 40 → 16 → 98 → 61 → 161 → 119 → 72 → 22 → 106 → 70
Route 7, cost: 3542, demand: 517:0 → 4 → 92 → 182 → 195 → 31 → 45 → 85 → 165 → 177 → 128 → 168 → 65 → 99 → 94 → 53 → 151 → 160 → 112 → 125 → 189 → 109 → 101 → 192 → 50 → 33 → 157
Route 8, cost: 3135, demand: 310:0 → 84 → 199 → 127 → 118 → 105 → 69 → 176 → 113 → 58 → 134 → 191 → 71 → 144 → 137 → 123 → 115 → 121 → 17 → 174
Route 9, cost: 2648, demand: 214:0 → 140 → 54 → 155 → 88 → 87 → 11 → 13 → 8 → 1 → 60 → 75 → 64
Route 10, cost: 2383, demand: 131:0 → 153 → 47 → 200 → 103 → 110 → 44 → 95 → 90 → 178 → 126
total cost =2992.3, Total routes = 11 found in iteration 13449 after 9 seconds
../../../data/Nsolomon/RC2_2_1.TXT&2992.3&11&16.247

B.10 RC2_2_2

Route 0, cost: 3543, demand: 471:0 → 153 → 47 → 48 → 131 → 124 → 43 → 180 → 3 → 136 → 187 → 130 → 174 → 17 → 87 → 88 → 71 → 11 → 13 → 8 → 60 → 1 → 76 → 34 → 2 → 141 → 185 → 175
Route 1, cost: 4135, demand: 346:0 → 42 → 172 → 197 → 181 → 35 → 164 → 104 → 25 → 19 → 170 → 103 → 80 → 27 → 51 → 145 → 91 → 126 → 178 → 83 → 110 → 36
Route 2, cost: 2737, demand: 446:0 → 4 → 92 → 182 → 57 → 133 → 40 → 147 → 31 → 117 → 108 → 38 → 32 → 23 → 86 → 114 → 74 → 52 → 106 → 22 → 70 → 195 → 158
Route 3, cost: 2574, demand: 426:0 → 150 → 163 → 28 → 6 → 49 → 14 → 63 → 149 → 120 → 135 → 159 → 5 → 62 → 26 → 20 → 46 → 138 → 186 → 122 → 77 → 10 → 73 → 193 → 146 → 93 → 39
Route 4, cost: 2398, demand: 318:0 → 142 → 152 → 167 → 169 → 98 → 16 → 119 → 161 → 72 → 129 → 41 → 194 → 55 → 107 → 139 → 66 → 89
Route 5, cost: 2826, demand: 275:0 → 64 → 84 → 199 → 127 → 118 → 105 → 176 → 113 → 58 → 134 → 191 → 69 → 144 → 137 → 123 → 75 → 97
Route 6, cost: 2906, demand: 457:0 → 154 → 81 → 165 → 85 → 45 → 56 → 177 → 65 → 128 → 168 → 99 → 94 → 151 → 53 → 160 → 112 → 125 → 189 → 109 → 101 → 192 → 50 → 157 → 33 → 173
Route 7, cost: 955, demand: 176:0 → 100 → 37 → 9 → 12 → 59 → 143 → 188 → 29 → 21 → 166 → 82
Route 8, cost: 3150, demand: 436:0 → 24 → 156 → 102 → 15 → 179 → 7 → 68 → 18 → 183 → 196 → 184 → 190 → 61 → 198 → 111 → 148 → 116 → 95 → 44 → 90 → 200 → 79 → 162 → 96
Route 9, cost: 1471, demand: 157:0 → 140 → 54 → 155 → 115 → 121 → 30 → 67
Route 10, cost: 446, demand: 50:0 → 132 → 78 → 171
total cost =2714.1, Total routes = 11 found in iteration 19584 after 14 seconds
../../../data/Nsolomon/RC2_2_2.TXT&2714.1&11&17.133

B.11 RC2_2_3

Route 0, cost: 6319, demand: 921:0 → 28 → 6 → 132 → 78 → 171 → 187 → 150 → 92 → 154 → 166 → 21 → 173 → 86 → 114 → 52 → 74 → 101 → 53 → 192 → 56 → 45 → 81 → 194 → 41 → 119 → 161 → 126 → 183 → 15 → 110 → 178 → 90 → 103 → 200 → 73 → 93 → 146 → 193 → 181 → 141 → 185 → 2 → 1 → 60 → 75 → 134 → 30 → 64 → 67
Route 1, cost: 5846, demand: 592:0 → 140 → 54 → 100 → 37 → 155 → 59 → 143 → 195 → 40 → 16 → 98 → 147 → 108 → 117 → 23 → 38 → 32 → 112 → 109 → 189 → 125 → 65 → 144 → 71 → 137 → 123 → 115 → 121 → 11 → 118 → 13 → 8 → 76 → 34 → 175 → 174
Route 2, cost: 5103, demand: 780:0 → 4 → 182 → 57 → 133 → 106 → 31 → 85 → 50 → 160 → 151 → 94 → 99 → 168 → 128 → 177 → 33 → 188 → 29 → 72 → 136 → 102 → 196 → 27 → 91 → 111 → 95 → 83 → 44 → 116 → 148 → 145 → 51 → 18 → 179 → 36 → 3 → 47 → 39 → 79 → 49 → 153
Route 3, cost: 2493, demand: 476:0 → 163 → 14 → 63 → 149 → 120 → 135 → 197 → 5 → 62 → 122 → 35 → 164 → 104 → 25 → 170 → 19 → 10 → 77 → 26 → 159 → 20 → 186 → 138 → 46 → 172 → 42 → 96 → 162 → 130
Route 4, cost: 2683, demand: 371:0 → 97 → 84 → 199 → 127 → 105 → 69 → 176 → 113 → 58 → 191 → 88 → 87 → 17 → 66 → 139 → 107 → 165 → 157 → 12 → 9 → 89
Route 5, cost: 3193, demand: 418:0 → 142 → 48 → 131 → 156 → 167 → 152 → 169 → 24 → 180 → 43 → 124 → 7 → 68 → 80 → 190 → 184 → 61 → 198 → 70 → 22 → 129 → 55 → 158 → 82
total cost =2563.7, Total routes = 6 found in iteration 24813 after 18 seconds
../../../data/Nsolomon/RC2_2_3.TXT&2563.7&6&18.271

B.12 RC2_2_5

Route 0, cost: 5923, demand: 585:0 → 100 → 9 → 12 → 59 → 143 → 173 → 177 → 165 → 85 → 45 → 56 → 108 → 114 → 86 → 166 → 87 → 144 → 123 → 137 → 71 → 11 → 115 → 121 → 17 → 174 → 96 → 34 → 76 → 2 → 185 → 141 → 193 → 19 → 146 → 39
Route 1, cost: 3800, demand: 475:0 → 24 → 48 → 131 → 156 → 102 → 15 → 179 → 68 → 7 → 200 → 63 → 149 → 186 → 20 → 159 → 26 → 170 → 103 → 90 → 110 → 44 → 95 → 83 → 148 → 36
Route 2, cost: 2579, demand: 357:0 → 163 → 162 → 42 → 172 → 197 → 5 → 181 → 35 → 164 → 104 → 25 → 10 → 77 → 122 → 62 → 46 → 138 → 135 → 73 → 93 → 120 → 175
Route 3, cost: 3033, demand: 502:0 → 89 → 37 → 21 → 139 → 29 → 188 → 65 → 128 → 99 → 94 → 168 → 53 → 101 → 74 → 52 → 106 → 129 → 41 → 158 → 194 → 55 → 81 → 107 → 82 → 154
Route 4, cost: 3131, demand: 448:0 → 140 → 54 → 155 → 88 → 84 → 199 → 127 → 118 → 105 → 69 → 191 → 58 → 134 → 113 → 176 → 13 → 8 → 1 → 60 → 75 → 97 → 30 → 64 → 67
Route 5, cost: 3215, demand: 452:0 → 153 → 49 → 14 → 47 → 136 → 3 → 180 → 43 → 124 → 18 → 183 → 196 → 184 → 190 → 27 → 51 → 61 → 198 → 111 → 91 → 145 → 116 → 80 → 178 → 79
Route 6, cost: 2614, demand: 339:0 → 4 → 92 → 195 → 31 → 23 → 117 → 32 → 38 → 112 → 160 → 109 → 189 → 125 → 151 → 192 → 50 → 33 → 157 → 66
Route 7, cost: 489, demand: 120:0 → 150 → 28 → 6 → 132 → 78 → 171 → 130 → 187
Route 8, cost: 1162, demand: 83:0 → 142 → 152 → 167 → 169 → 72
Route 9, cost: 1625, demand: 197:0 → 182 → 57 → 133 → 147 → 40 → 16 → 98 → 119 → 161 → 22 → 70 → 126
total cost =2757.1, Total routes = 10 found in iteration 23876 after 15 seconds
../../../data/Nsolomon/RC2_2_5.TXT&2757.1&10&16.36

B.13 RC2_2_6

Route 0, cost: 3752, demand: 601:0 → 100 → 37 → 9 → 12 → 59 → 143 → 173 → 177 → 65 → 128 → 168 → 99 → 94 → 151 → 160 → 53 → 32 → 38 → 114 → 86 → 117 → 23 → 52 → 74 → 112 → 125 → 189 → 109 → 192 → 101 → 50 → 33 → 157 → 66
Route 1, cost: 3767, demand: 650:0 → 153 → 28 → 163 → 6 → 49 → 47 → 162 → 63 → 149 → 42 → 172 → 197 → 5 → 62 → 181 → 35 → 164 → 104 → 25 → 170 → 10 → 77 → 122 → 26 → 20 → 159 → 46 → 138 → 186 → 135 → 93 → 73 → 141 → 185 → 2 → 76 → 34 → 175
Route 2, cost: 4818, demand: 700:0 → 142 → 24 → 131 → 48 → 124 → 156 → 102 → 15 → 179 → 18 → 68 → 7 → 103 → 200 → 14 → 132 → 78 → 130 → 69 → 105 → 176 → 113 → 134 → 58 → 191 → 87 → 71 → 144 → 123 → 137 → 13 → 8 → 1 → 60 → 75 → 97 → 30 → 64 → 67
Route 3, cost: 4087, demand: 549:0 → 4 → 92 → 182 → 57 → 133 → 195 → 165 → 85 → 45 → 56 → 108 → 31 → 147 → 40 → 16 → 98 → 119 → 161 → 72 → 129 → 41 → 22 → 106 → 70 → 126 → 148 → 83
Route 4, cost: 4035, demand: 425:0 → 169 → 152 → 167 → 190 → 183 → 43 → 3 → 136 → 180 → 196 → 184 → 27 → 51 → 61 → 198 → 111 → 95 → 44 → 90 → 110 → 178 → 116 → 145 → 91 → 80
Route 5, cost: 2318, demand: 190:0 → 84 → 199 → 127 → 118 → 88 → 155 → 115 → 121 → 11 → 17 → 174
Route 6, cost: 2462, demand: 161:0 → 150 → 187 → 171 → 79 → 96 → 120 → 193 → 19 → 146 → 39 → 36
Route 7, cost: 1604, demand: 282:0 → 140 → 54 → 89 → 139 → 188 → 29 → 21 → 166 → 107 → 81 → 55 → 194 → 158 → 82 → 154
total cost =2684.3, Total routes = 8 found in iteration 23168 after 15 seconds
../../../data/Nsolomon/RC2_2_6.TXT&2684.3&8&15.833

Appendix C

Homberger solutions

In this Appendix I have listed the results of the test on the Homberger [5] instances. The instances are divided such that the tests with vehicle capacity 200 is in one table for each customer size, and the tests with vehicle capacity 700 is in another table.

The 800 customer instances could not be divided by the Voronoi Algorithm with centers found by Kruskal for the Randomly distributed customers, or the random and clustered instances. Thus the only test on those are without division of the problem.

C.1 Homberger 200

Table C.1: Results for the 200 customer Homberger instances with vehicle capacity 200.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C1_2_1.TXT	2823.2	20	14.206	2704.57	20
C1_2_2.TXT	2879.3	21	14.51	2917,89	18
C1_2_3.TXT	2866.2	20	15.549	2707,35	18
C1_2_4.TXT	3017.6	19	15.77	2643,31	18
C1_2_5.TXT	2763.1	20	14.501	2702,05	20
C1_2_6.TXT	2817.9	20	14.244	2701,04	20
C1_2_7.TXT	2768.5	20	14.549	2701,04	20
C1_2_8.TXT	2966.2	21	14.431	2769,19	18
C1_2_9.TXT	2867	19	14.591	2642,82	18
C1_210.TXT	2923.9	21	15.212	2643,51	18
R1_2_1.TXT	4965.7	24	16.023	5024,65	19
R1_2_2.TXT	4132.6	22	15.606	4040,60	18
R1_2_3.TXT	3620.3	19	15.013	3381,96	18
R1_2_4.TXT	3424.5	19	14.435	3059,22	18
R1_2_5.TXT	4326.9	21	15.014	4107,86	18
R1_2_6.TXT	3723.3	20	14.417	3583,14	18
R1_2_7.TXT	3491.4	19	14.092	3150,11	18
R1_2_8.TXT	3410.6	18	13.213	2952,65	18
R1_2_9.TXT	4077.2	19	14.096	3760,58	18
R1_210.TXT	3619.3	19	13.667	3301,18	18
RC1_2_1.TXT	3724.7	21	14.229	3602,80	18
RC1_2_2.TXT	3355.2	19	14.272	3249,50	18
RC1_2_3.TXT	3237	19	13.998	3008,33	18
RC1_2_4.TXT	3153	19	13.523	2852,62	18
RC1_2_5.TXT	3545.8	20	14.734	3385,88	18
RC1_2_6.TXT	3643.9	20	14.297	3324,80	18
RC1_2_7.TXT	3427.3	19	13.493	3189,32	18
RC1_2_8.TXT	3294	19	13.81	3083,93	18
RC1_2_9.TXT	3317.4	19	13.387	3083,41	18
RC1_210.TXT	3281.2	19	13.541	3008,53	18

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

Table C.2: Results for the 200 customer Homberger instances with vehicle capacity 700.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C2_2_1.TXT	2064.2	9	14.099	1931,44	6
C2_2_2.TXT	2086.3	10	16.389	1863,16	6
C2_2_3.TXT	1955.4	7	18.689	1775,11	6
C2_2_4.TXT	1879.5	8	21.51	1703,43	6
C2_2_5.TXT	1989.6	10	15.001	1878,85	6
C2_2_6.TXT	2040.8	8	16.277	1857,35	6
C2_2_7.TXT	2000.9	9	16.204	1849,46	6
C2_2_8.TXT	1916.7	9	16.935	1820,53	6
C2_2_9.TXT	2121	10	17.593	1830,05	6
C2_210.TXT	1934.4	8	18.431	1806,60	6
R2_2_1.TXT	3619.2	14	15.813	4483,16	4
R2_2_2.TXT	3287.6	13	17.217	3621,20	4
R2_2_3.TXT	3124.3	5	18.717	2880,62	4
R2_2_4.TXT	2350.4	4	19.43	1981,29	4
R2_2_5.TXT	3651	5	15.457	3366,79	4
R2_2_6.TXT	3296.8	5	17.041	2913,03	4
R2_2_7.TXT	2779.4	5	18.336	2451,14	4
R2_2_8.TXT	2134.6	4	20.54	1849,87	4
R2_2_9.TXT	3276.1	5	15.681	3092,53	4
R2_210.TXT	3092.5	4	15.518	2654,97	4
RC2_2_1.TXT	2980.1	13	15.975	3099,53	6
RC2_2_2.TXT	2714.1	11	17.133	2825,24	5
RC2_2_3.TXT	2553.9	7	17.972	2603,83	4
RC2_2_4.TXT	2314.7	5	18.818	2043,05	4
RC2_2_5.TXT	2685.5	10	16.208	2911,46	4
RC2_2_6.TXT	2684.3	8	15.833	2975,13	4
RC2_2_7.TXT	2855	5	16.78	2525,83	4
RC2_2_8.TXT	2641.3	5	17.781	2293,35	4
RC2_2_9.TXT	2395.6	8	19.411	2175,04	4
RC2_210.TXT	2550.4	4	21.184	2015,60	4

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

C.2 Homberger 400

Table C.3: Results for the 400 customer Homberger instances with vehicle capacity 200.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C1_4_1.TXT	9156.8	47	32.3	7152,02	40
C1_4_2.TXT	7945.9	43	30.508	7687,38	36
C1_4_3.TXT	8267	39	30.09	7060,73	36
C1_4_4.TXT	8791.1	40	32.584	6816,17	36
C1_4_5.TXT	8598.1	44	30.935	7152,02	40
C1_4_6.TXT	8575.5	44	30.457	7153,41	40
C1_4_7.TXT	8399.2	43	29.372	7417,92	39
C1_4_8.TXT	8773.7	42	30.992	7363,51	37
C1_4_9.TXT	8645.2	40	30.888	7061,21	36
C1_410.TXT	8187.9	38	30.018	6860,63	36
R1_4_1.TXT	11737.3	45	31.195	11084,00	38
R1_4_2.TXT	10353	40	29.909	8955,50	36
R1_4_3.TXT	9526.8	37	27.115	7841,52	36
R1_4_4.TXT	9026.4	37	25.735	7318,62	36
R1_4_5.TXT	9953.1	39	29.646	9242,43	36
R1_4_6.TXT	9793.1	38	27.334	8383,67	36
R1_4_7.TXT	9426.9	37	25.233	7656,94	36
R1_4_8.TXT	8978.3	37	25.202	7293,69	36
R1_4_9.TXT	9969	38	27.7	8750,84	36
R1_410.TXT	9593.4	37	25.803	8125,03	36
RC1_4_1.TXT	9452.5	41	29.098	8630,94	36
RC1_4_2.TXT	9160.5	40	29.45	7958,67	36
RC1_4_3.TXT	9010.2	37	26.881	7562,60	36
RC1_4_4.TXT	9173.9	37	25.179	7332,59	36
RC1_4_5.TXT	9090.8	39	27.857	8249,63	36
RC1_4_6.TXT	9534.9	40	27.918	8223,12	36
RC1_4_7.TXT	9167.2	38	27.286	8001,12	36
RC1_4_8.TXT	9011.8	38	26.121	7836,29	36
RC1_4_9.TXT	8973.6	37	25.968	7811,55	36
RC1_410.TXT	9141.9	37	25.586	7668,77	36

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

Table C.4: Results for the 400 customer Homberger instances with vehicle capacity 700.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C2_4_1.TXT	4648.1	17	30.161	4116,05	12
C2_4_2.TXT	4646.3	15	33.119	3929,89	12
C2_4_3.TXT	4406.1	14	37.14	4109,88	11
C2_4_4.TXT	4337.8	16	43.225	3865,45	11
C2_4_5.TXT	4689.6	18	32.024	3938,69	12
C2_4_6.TXT	4724	19	33.137	3875,94	12
C2_4_7.TXT	4651.6	17	34.38	3894,13	12
C2_4_8.TXT	4513.6	15	33.924	3787,08	12
C2_4_9.TXT	4494.9	15	34.948	3865,65	12
C2_410.TXT	4763.9	16	36.459	4115,46	11
R2_4_1.TXT	8150.9	27	34.14	9213,68	8
R2_4_2.TXT	8400.7	13	33.45	7606,75	8
R2_4_3.TXT	7014.4	10	32.085	5988,02	8
R2_4_4.TXT	5924.6	8	31.109	4297,20	8
R2_4_5.TXT	8956.2	8	28.599	7143,55	8
R2_4_6.TXT	7970.2	8	27.21	6163,81	8
R2_4_7.TXT	6771.8	8	28.809	5082,10	8
R2_4_8.TXT	5671.5	8	30.426	4051,98	8
R2_4_9.TXT	7749.1	9	29.285	6493,13	8
R2_410.TXT	7354.1	8	27.545	5844,77	8
RC2_4_1.TXT	6826.7	23	32.507	6688,31	11
RC2_4_2.TXT	6311.5	20	34.867	6355,59	9
RC2_4_3.TXT	6136.2	9	34.654	4958,74	8
RC2_4_4.TXT	5568.3	8	29.557	3635,04	8
RC2_4_5.TXT	6465.6	21	32.815	5923,95	9
RC2_4_6.TXT	6622.9	24	33.537	5863,56	8
RC2_4_7.TXT	5968	16	33.534	5466,70	8
RC2_4_8.TXT	6133.3	9	32.356	4848,87	8
RC2_4_9.TXT	6148.2	8	32.468	4599,57	8
RC2_410.TXT	5441.9	8	34.202	4311,59	8

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

C.3 Homberger 600

Table C.5: Results for the 600 customer Homberger instances with vehicle capacity 200.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C1_6_1.TXT	18165.4	72	51.937	14095,64	60
C1_6_2.TXT	18251.9	69	52.372	14163,31	56
C1_6_3.TXT	18246.3	64	52.243	13781,19	56
C1_6_4.TXT	17272.7	58	49.1	13571,88	56
C1_6_5.TXT	17783.4	68	51.721	14085,70	60
C1_6_6.TXT	18538.6	72	52.109	14089,56	60
C1_6_7.TXT	18135.4	67	51.89	14851,65	58
C1_6_8.TXT	17809.9	66	52.213	14541,53	56
C1_6_9.TXT	18088.8	63	51.471	13718,23	56
C1_610.TXT	17755.8	61	51.288	13669,88	56
R1_6_1.TXT	24611.6	65	52.691	21131,09	59
R1_6_2.TXT	21816.1	59	49.063	19147,38	54
R1_6_3.TXT	20837.3	56	47.044	17216,16	54
R1_6_4.TXT	19548.6	56	39.343	15947,03	54
R1_6_5.TXT	22556.5	62	49.409	20017,80	54
R1_6_6.TXT	22292.8	56	46.719	18237,76	54
R1_6_7.TXT	20766.2	56	44.713	16796,63	54
R1_6_8.TXT	19921.7	55	37.653	15725,86	54
R1_6_9.TXT	21403.8	56	47.715	19015,51	54
R1_610.TXT	21531.7	56	43.342	18204,18	54
RC1_6_1.TXT	19656	62	95.778	17317,13	55
RC1_6_2.TXT	18994.9	59	93.242	16123,40	55
RC1_6_3.TXT	18856.5	57	87.036	15358,13	55
RC1_6_4.TXT	18837.3	56	73.223	14872,79	55
RC1_6_5.TXT	19649	60	92.315	16934,45	55
RC1_6_6.TXT	19355.6	59	89.825	16842,27	55
RC1_6_7.TXT	19373.8	58	69.608	16450,42	55
RC1_6_8.TXT	19335.7	57	44.034	16164,82	55
RC1_6_9.TXT	19318.8	58	43.917	16153,00	55
RC1_610.TXT	18553.8	57	44.511	15936,81	55

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

Table C.6: Results for the 600 customer Homberger instances with vehicle capacity 700.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C2_6_1.TXT	9620.7	27	56.812	7774,10	18
C2_6_2.TXT	9562.6	27	52.904	8380,49	17
C2_6_3.TXT	9213.6	26	58.158	7595,43	17
C2_6_4.TXT	8705.1	23	66.442	6993,77	17
C2_6_5.TXT	9730	28	50.481	7575,20	18
C2_6_6.TXT	9948.9	29	51.46	7472,24	18
C2_6_7.TXT	9910.1	27	52.468	7512,33	18
C2_6_8.TXT	9725.1	28	53.385	7778,30	17
C2_6_9.TXT	9080.6	25	54.428	7350,94	18
C2_610.TXT	9572.6	29	57.074	7523,34	17
R2_6_1.TXT	16796.7	36	53.208	18291,18	11
R2_6_2.TXT	16022.3	35	54.467	14995,76	11
R2_6_3.TXT	13621	19	53.743	11255,49	11
R2_6_4.TXT	10979.4	18	53.201	8126,87	11
R2_6_5.TXT	16956.7	29	48.331	15357,25	11
R2_6_6.TXT	14971.6	28	51.278	12803,83	11
R2_6_7.TXT	13342.3	14	49.963	10172,17	11
R2_6_8.TXT	10373.5	14	52.278	7752,78	11
R2_6_9.TXT	17000	39	95.695	13567,84	11
R2_610.TXT	14940	20	90.834	12513,45	11
RC2_6_1.TXT	13539.3	31	51.485	13163,03	15
RC2_6_2.TXT	12852.7	32	53.099	11853,72	12
RC2_6_3.TXT	11912.9	25	54.683	9816,47	11
RC2_6_4.TXT	9786.6	14	50.75	7197,11	11
RC2_6_5.TXT	13879.2	28	50.649	12168,79	12
RC2_6_6.TXT	13734.4	32	50.754	12282,52	11
RC2_6_7.TXT	13952.9	32	51.264	10929,56	11
RC2_6_8.TXT	12703.6	24	51.571	10474,95	11
RC2_6_9.TXT	12180.8	17	49.516	9821,39	11
RC2_610.TXT	11281.6	22	52.351	9339,41	11

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

C.4 Homberger 800

Table C.7: Results for the 800 customer Homberger instances with vehicle capacity 200.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C1_8_1.TXT	34376.6	89	76.13	25030,36	80
C1_8_2.TXT	36158.9	100	75.65	25528,55	74
C1_8_3.TXT	31753.7	79	69.47	24366,83	72
C1_8_4.TXT	31623.3	80	60.77	23917,70	72
C1_8_5.TXT	34470.6	86	75.88	25166,28	80
C1_8_6.TXT	37627.9	106	77.56	25160,83	80
C1_8_7.TXT	35687.4	102	77.44	26639,13	77
C1_8_8.TXT	34860.9	98	69.67	25370,02	74
C1_8_9.TXT	33766.3	92	68.01	24698,05	72
C1_810.TXT	33644.4	85	69.34	24324,76	72
R1_8_1.TXT	44910.6	95	78.88	39612,2	79
R1_8_2.TXT	41773.5	93	70.84	33190,68	72
R1_8_3.TXT	40691.4	89	60.7	29943,87	72
R1_8_4.TXT	37627.8	73	50.74	26838,04	72
R1_8_5.TXT	40977.1	92	73.58	34247,99	72
R1_8_6.TXT	39907.8	85	68.18	31728,99	72
R1_8_7.TXT	40954.3	77	60.42	29399,21	72
R1_8_8.TXT	38571.5	73	50.17	28191,89	72
R1_8_9.TXT	40079.1	86	64.1	33074,30	72
R1_810.TXT	39221	83	62.62	31730,45	72
RC1_8_1.TXT	37833.8	90	68.68	35102,79	72
RC1_8_2.TXT	37186.8	87	64.64	33361,67	72
RC1_8_3.TXT	37977.4	79	54.41	30608,16	72
RC1_8_4.TXT	34745	79	52.55	28363,65	72
RC1_8_5.TXT	36939.8	88	65.66	34481,02	72
RC1_8_6.TXT	36017.5	85	64.17	34849,96	72
RC1_8_7.TXT	36277.2	85	60.19	33102,75	72
RC1_8_8.TXT	37086	84	57.43	33188,75	72
RC1_8_9.TXT	35854.5	84	56.01	33350,51	72
RC1_810.TXT	36427	79	52.5	31766,56	72

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

Table C.8: Results for the 800 customer Homberger instances with vehicle capacity 700.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C2_8_1.TXT	15246.9	34	62.2	11654,81	24
C2_8_2.TXT	16741.6	41	69.15	12332,37	23
C2_8_3.TXT	20666.8	44	76.98	11438,72	23
C2_8_4.TXT	17525.4	35	89.04	10963,49	23
C2_8_5.TXT	17424.3	45	69.51	11432,92	24
C2_8_6.TXT	16791.9	42	67.39	11357,86	24
C2_8_7.TXT	17807.5	43	68.51	11378,67	24
C2_8_8.TXT	19151.2	43	69.81	12927,45	23
C2_8_9.TXT	15986.9	36	69.67	12301,63	23
C2_810.TXT	15615.4	31	72.07	11163,89	23
R2_8_1.TXT	39005.8	56	75.23	28392,87	15
R2_8_2.TXT	37031	47	66.01	23274,22	15
R2_8_3.TXT	29893.3	51	60.98	17992,25	15
R2_8_4.TXT	23610.3	44	58.35	13413,79	15
R2_8_5.TXT	37888.3	59	61.16	24611,39	15
R2_8_6.TXT	31358.4	62	57.49	20697,06	15
R2_8_7.TXT	23594	38	56.82	16977,49	15
R2_8_8.TXT	22351.7	15	57.13	12945,52	15
R2_8_9.TXT	34968.5	63	58.53	22588,02	15
R2_810.TXT	30473.4	29	49.86	21092,27	15
RC2_8_1.TXT	31963.1	39	70.3	20520,49	19
RC2_8_2.TXT	27608.7	28	65.09	18032,89	17
RC2_8_3.TXT	27809.8	21	67.9	14800,78	15
RC2_8_4.TXT	20656.4	21	60.64	11312,68	15
RC2_8_5.TXT	28020.8	29	64.45	18917,65	16
RC2_8_6.TXT	30250.8	23	64.94	18600,22	15
RC2_8_7.TXT	25972.5	46	63.75	17327,53	15
RC2_8_8.TXT	24772	24	56.7	16203,18	15
RC2_8_9.TXT	21935.8	29	55.45	15622,52	15
RC2_810.TXT	28831.2	53	56.36	14892,29	15

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

C.5 Homberger 1000

Table C.9: Results for the 1000 customer Homberger instances with vehicle capacity 200.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C110_1.TXT	59949.4	129	88.405	42478,95	100
C110_2.TXT	53184.2	113	78.511	42242,95	91
C110_3.TXT	50168.3	102	74.579	40376,43	90
C110_4.TXT	52896	92	62.517	39735,30	90
C110_5.TXT	57510.6	121	87.045	42469,18	100
C110_6.TXT	54595.3	120	87.907	42470,04	100
C110_7.TXT	53886.5	114	81.981	42824,09	98
C110_8.TXT	53903	113	79.513	42499,59	93
C110_9.TXT	52127.1	106	77.241	41318,12	90
C11010.TXT	52263.3	101	77.373	40586,60	90
R110_1.TXT	63796.1	100	68.109	53904,23	100
R110_2.TXT	60691.7	95	64.945	50701,78	91
R110_3.TXT	59333.2	94	62.964	46169,17	91
R110_4.TXT	58719.7	101	60.265	43461,84	91
R110_5.TXT	61268.3	95	63.487	54032,44	91
R110_6.TXT	61049	95	61.749	49059,80	91
R110_7.TXT	60562.6	93	59.296	45729,79	91
R110_8.TXT	58123.4	97	59.486	42767,77	91
R110_9.TXT	59764.4	99	64.962	51391,80	91
R11010.TXT	60781.4	117	63.749	49348,36	91
RC110_1.TXT	54189.5	103	73.12	47143,90	90
RC110_2.TXT	52157.3	96	69.158	44906,58	90
RC110_3.TXT	56198.1	92	62.687	43390,58	90
RC110_4.TXT	53393.8	96	59.482	41917,14	90
RC110_5.TXT	53685.9	97	64.078	46631,89	90
RC110_6.TXT	54269.3	94	63.337	46391,60	90
RC110_7.TXT	54233.2	93	60.68	46157,71	90
RC110_8.TXT	57344.4	97	58.76	45406,46	90
RC110_9.TXT	56750.1	96	57.427	45149,72	90
RC11010.TXT	55669.5	93	57.482	44947,71	90

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

Table C.10: Results for the 1000 customer Homberger instances with vehicle capacity 700.

instance	result	# Vehicles	time (s)	best res	# Vehicles
C210_1.TXT	22173.2	40	75.479	16879,24	30
C210_2.TXT	22610.7	41	83.608	17144,29	29
C210_3.TXT	20335.8	37	87.454	16367,59	29
C210_4.TXT	19810.5	34	105.375	15919,46	29
C210_5.TXT	22485.4	41	78.563	16561,70	30
C210_6.TXT	21769.3	42	81.875	16341,67	30
C210_7.TXT	21834.8	41	81.737	16435,10	30
C210_8.TXT	21204	40	84.919	16315,89	29
C210_9.TXT	21875.7	40	86.827	16751,82	29
C21010.TXT	21088.3	36	83.42	15885,41	29
R210_1.TXT	46758.3	73	90.724	42467,87	19
R210_2.TXT	38125.9	42	91.324	33589,08	19
R210_3.TXT	39298	19	69.295	25321,00	19
R210_4.TXT	31133.1	32	64.772	18222,30	19
R210_5.TXT	47671.6	37	79.057	36735,20	19
R210_6.TXT	42013.4	20	68.583	30261,75	19
R210_7.TXT	38108.3	36	62.869	23463,80	19
R210_8.TXT	35166.8	19	64.938	17705,20	19
R210_9.TXT	45675.8	22	68.869	33519,84	19
R21010.TXT	46516.9	19	65.659	30706,00	19
RC210_1.TXT	34446.9	47	82.397	29754,06	21
RC210_2.TXT	32857.7	46	85.819	27552,05	18
RC210_3.TXT	28336.8	23	83.27	20276,16	18
RC210_4.TXT	30329.8	32	67.384	15954,60	18
RC210_5.TXT	37504.6	64	86.698	27766,56	18
RC210_6.TXT	39109.4	23	77.764	27003,30	18
RC210_7.TXT	35504.8	47	83.434	25526,73	18
RC210_8.TXT	35977.8	19	62.777	24335,40	18
RC210_9.TXT	36435.9	19	63.482	23465,51	18
RC21010.TXT	34968.3	18	59.469	22481,03	18

The columns are: the instance, my best result, the number of vehicles used in the solution and the time used to solve it. The last two columns are the best known solutions.

Appendix D

Real World Routes, Day 2

Actual time, real distance

Route 0, cost: 2857, demand: 33:0 → 398 → 396 → 369 → 448 → 370
Route 1, cost: 4550, demand: 24:0 → 305 → 461 → 311 → 367 → 324
Route 2, cost: 2330, demand: 33:0 → 353 → 335 → 268 → 436 → 301
Route 3, cost: 4566, demand: 33:0 → 405 → 429 → 377 → 413 → 416 → 418
Route 4, cost: 2401, demand: 32:0 → 279 → 264 → 427 → 291 → 296
Route 5, cost: 3953, demand: 33:0 → 432 → 340 → 434 → 259 → 450 → 421
Route 6, cost: 3115, demand: 33:0 → 344 → 447 → 375 → 399 → 314
Route 7, cost: 1737, demand: 28:0 → 313 → 365 → 288 → 358 → 384
Route 8, cost: 2304, demand: 28:0 → 454 → 439 → 443 → 286
Route 9, cost: 2956, demand: 31:0 → 455 → 404 → 409 → 290
Route 10, cost: 3145, demand: 32:0 → 320 → 274 → 355 → 336 → 342 → 275
Route 11, cost: 1660, demand: 29:0 → 257 → 386 → 348 → 285 → 423 → 309
Route 12, cost: 3315, demand: 33:0 → 282 → 283 → 272 → 363 → 397 → 323 → 426
Route 13, cost: 1983, demand: 31:0 → 273 → 262 → 406 → 276 → 437
Route 14, cost: 5481, demand: 27:0 → 402 → 407 → 332 → 295 → 364
Route 15, cost: 2030, demand: 29:0 → 403 → 352 → 428 → 383 → 376
Route 16, cost: 2569, demand: 28:0 → 280 → 250 → 362 → 303 → 431 → 445 → 297 → 293
Route 17, cost: 2377, demand: 27:0 → 312 → 387 → 270 → 333 → 425
Route 18, cost: 1691, demand: 29:0 → 256 → 361 → 453 → 329
Route 19, cost: 5483, demand: 32:0 → 310 → 385 → 456 → 315 → 391 → 339
Route 20, cost: 1602, demand: 30:0 → 459 → 331 → 389 → 382

Route 21, cost: 4401, demand: 31:0 → 321 → 278 → 433 → 307 → 287 → 308 → 346
Route 22, cost: 2707, demand: 33:0 → 318 → 281 → 415 → 284 → 325 → 328
Route 23, cost: 3214, demand: 30:0 → 457 → 345 → 379 → 420
Route 24, cost: 790, demand: 25:0 → 359 → 271 → 265 → 334 → 306 → 266
Route 25, cost: 2190, demand: 30:0 → 446 → 248 → 343 → 277
Route 26, cost: 2479, demand: 31:0 → 424 → 294 → 292 → 400 → 300 → 255
Route 27, cost: 2382, demand: 26:0 → 356 → 249 → 451 → 366 → 393 → 374
Route 28, cost: 3106, demand: 33:0 → 258 → 322 → 460 → 302 → 412
Route 29, cost: 2417, demand: 25:0 → 254 → 299 → 444 → 316
Route 30, cost: 843, demand: 28:0 → 422 → 441 → 380 → 449
Route 31, cost: 2615, demand: 29:0 → 317 → 354 → 351 → 304 → 360 → 392
Route 32, cost: 2088, demand: 29:0 → 263 → 289 → 430 → 373
Route 33, cost: 2851, demand: 33:0 → 350 → 381 → 326 → 378 → 327 → 261 → 347
Route 34, cost: 2207, demand: 32:0 → 452 → 298 → 410 → 390 → 253 → 341
Route 35, cost: 1823, demand: 22:0 → 338 → 417 → 372 → 395
Route 36, cost: 2443, demand: 32:0 → 357 → 419 → 440 → 388 → 435
Route 37, cost: 856, demand: 32:0 → 267 → 251 → 438 → 349 → 319 → 368
Route 38, cost: 2401, demand: 29:0 → 330 → 337 → 401
Route 39, cost: 1383, demand: 31:0 → 414 → 408 → 371 → 442 → 411
Route 40, cost: 1352, demand: 33:0 → 260 → 394 → 458 → 252 → 269
total cost =10665.3, Total routes = 41
found in iteration 19644 after 13 seconds

65 km/h, real distance

Route 0, cost: 1895, demand: 30:0 → 179 → 217 → 36 → 82 → 124 → 5
Route 1, cost: 2174, demand: 28:0 → 23 → 235 → 86 → 182 → 87 → 240
Route 2, cost: 2252, demand: 31:0 → 163 → 165 → 72 → 105 → 63
Route 3, cost: 2749, demand: 22:0 → 170 → 49 → 31 → 233 → 18
Route 4, cost: 2886, demand: 29:0 → 8 → 24 → 178 → 68
Route 5, cost: 2380, demand: 32:0 → 97 → 56 → 21 → 225 → 158
Route 6, cost: 2582, demand: 30:0 → 69 → 243 → 98 → 205 → 204
Route 7, cost: 2024, demand: 29:0 → 168 → 176 → 226 → 35
Route 8, cost: 2464, demand: 31:0 → 159 → 27 → 246 → 227 → 74
Route 9, cost: 5060, demand: 31:0 → 185 → 208 → 102 → 169 → 207 → 10
Route 10, cost: 1744, demand: 33:0 → 123 → 108 → 166 → 73 → 197
Route 11, cost: 2249, demand: 33:0 → 112 → 59 → 184 → 1 → 174 → 151
Route 12, cost: 3011, demand: 30:0 → 42 → 119 → 114 → 55 → 157

```
Route 13, cost: 1659, demand: 33:0 → 40 → 209 → 64 → 26 → 76 → 43 → 19
Route 14, cost: 2568, demand: 32:0 → 66 → 44 → 244 → 41 → 3
Route 15, cost: 2973, demand: 33:0 → 191 → 85 → 104 → 61 → 92 → 135 → 175
Route 16, cost: 2081, demand: 27:0 → 17 → 153 → 230 → 121 → 94
Route 17, cost: 3552, demand: 33:0 → 247 → 154 → 210 → 52 → 14 → 53 → 34
Route 18, cost: 2209, demand: 32:0 → 148 → 128 → 223 → 120
Route 19, cost: 2815, demand: 33:0 → 80 → 16 → 167 → 9 → 25 → 232 → 130 →
136
Route 20, cost: 1920, demand: 33:0 → 96 → 222 → 181 → 228 → 89
Route 21, cost: 768, demand: 24:0 → 39 → 106 → 214 → 139
Route 22, cost: 1347, demand: 33:0 → 46 → 231 → 22 → 45 → 126 → 93
Route 23, cost: 3386, demand: 29:0 → 201 → 141 → 131 → 127 → 215
Route 24, cost: 1775, demand: 31:0 → 62 → 6 → 237 → 29 → 171 → 150
Route 25, cost: 2003, demand: 33:0 → 224 → 206 → 99 → 2 → 30 → 236 → 183
Route 26, cost: 293, demand: 26:0 → 71 → 101 → 48 → 203 → 245 → 84
Route 27, cost: 3445, demand: 33:0 → 160 → 195 → 221 → 107 → 145 → 212
Route 28, cost: 4373, demand: 32:0 → 90 → 194 → 4 → 115 → 202 → 177
Route 29, cost: 2572, demand: 33:0 → 234 → 192 → 28 → 199 → 15 → 117
Route 30, cost: 3464, demand: 32:0 → 187 → 241 → 38 → 242 → 111 → 164
Route 31, cost: 1976, demand: 33:0 → 125 → 60 → 54 → 91 → 37
Route 32, cost: 2760, demand: 31:0 → 149 → 103 → 196 → 146
Route 33, cost: 3202, demand: 32:0 → 189 → 75 → 134 → 79 → 7
Route 34, cost: 2074, demand: 30:0 → 213 → 77 → 129 → 113
Route 35, cost: 1812, demand: 31:0 → 152 → 132 → 95 → 118 → 229
Route 36, cost: 5788, demand: 33:0 → 138 → 180 → 198 → 70 → 67 → 161
Route 37, cost: 2441, demand: 31:0 → 88 → 219 → 116 → 238
Route 38, cost: 2374, demand: 33:0 → 193 → 220 → 57 → 47 → 156
Route 39, cost: 131, demand: 33:0 → 100 → 51 → 200 → 216 → 81
Route 40, cost: 2415, demand: 29:0 → 58 → 32 → 211 → 83 → 142 → 143
Route 41, cost: 1808, demand: 33:0 → 11 → 12 → 13 → 33 → 110 → 137
Route 42, cost: 3076, demand: 29:0 → 162 → 122 → 140 → 50 → 147
Route 43, cost: 493, demand: 20:0 → 155 → 78 → 218
Route 44, cost: 5712, demand: 27:0 → 20 → 144 → 172 → 239 → 186
Route 45, cost: 266, demand: 4:0 → 188
Route 46, cost: 2468, demand: 33:0 → 190 → 65 → 109 → 133 → 173
total cost =11546.9, Total routes = 47
found in iteration 24732 after 19 seconds
```

Actual time, Euclidean distance

Route 0, cost: 1867, demand: 29:0 → 254 → 424 → 294 → 292 → 400 → 395
Route 1, cost: 3986, demand: 33:0 → 305 → 307 → 287 → 308 → 367 → 324
Route 2, cost: 1827, demand: 33:0 → 386 → 348 → 363 → 397 → 268 → 436 → 301
Route 3, cost: 2679, demand: 33:0 → 344 → 274 → 355 → 336 → 342 → 346
Route 4, cost: 1739, demand: 25:0 → 318 → 415 → 264 → 442 → 368
Route 5, cost: 2073, demand: 32:0 → 279 → 280 → 250 → 258 → 276
Route 6, cost: 2373, demand: 33:0 → 398 → 396 → 409 → 369 → 322 → 293
Route 7, cost: 2347, demand: 31:0 → 452 → 270 → 362 → 381 → 375 → 304 → 275
Route 8, cost: 4518, demand: 27:0 → 402 → 407 → 332 → 295 → 364
Route 9, cost: 1930, demand: 32:0 → 298 → 357 → 333 → 303 → 431 → 302 → 297
Route 10, cost: 4081, demand: 31:0 → 320 → 377 → 385 → 413 → 418
Route 11, cost: 1310, demand: 30:0 → 422 → 441 → 335 → 285 → 423 → 323
Route 12, cost: 1698, demand: 33:0 → 281 → 390 → 419 → 425 → 427
Route 13, cost: 2732, demand: 33:0 → 321 → 278 → 433 → 461 → 311 → 269 → 449
Route 14, cost: 1212, demand: 26:0 → 256 → 439 → 361 → 265 → 306 → 393
Route 15, cost: 1052, demand: 30:0 → 414 → 408 → 458 → 411 → 296
Route 16, cost: 1245, demand: 32:0 → 417 → 372 → 288 → 299 → 444
Route 17, cost: 3807, demand: 33:0 → 429 → 310 → 456 → 315 → 391 → 392
Route 18, cost: 2469, demand: 32:0 → 317 → 447 → 404 → 378 → 261 → 290 → 370
Route 19, cost: 1350, demand: 29:0 → 263 → 289 → 430 → 373
Route 20, cost: 1933, demand: 33:0 → 252 → 435 → 399 → 314 → 412
Route 21, cost: 1552, demand: 33:0 → 359 → 443 → 331 → 389 → 382
Route 22, cost: 2072, demand: 32:0 → 267 → 251 → 291 → 360 → 445 → 401 → 339
Route 23, cost: 1120, demand: 32:0 → 353 → 249 → 451 → 343 → 277
Route 24, cost: 1428, demand: 33:0 → 454 → 356 → 446 → 248 → 286
Route 25, cost: 1302, demand: 29:0 → 260 → 365 → 358 → 316 → 384
Route 26, cost: 2972, demand: 31:0 → 405 → 340 → 434 → 259 → 450 → 255
Route 27, cost: 485, demand: 32:0 → 313 → 438 → 319 → 349 → 300
Route 28, cost: 2425, demand: 33:0 → 455 → 326 → 327 → 347 → 448
Route 29, cost: 1143, demand: 25:0 → 273 → 376 → 262 → 406
Route 30, cost: 1964, demand: 33:0 → 312 → 337 → 387 → 410 → 341
Route 31, cost: 1647, demand: 23:0 → 459 → 453 → 329
Route 32, cost: 2317, demand: 32:0 → 282 → 283 → 272 → 426 → 309
Route 33, cost: 1631, demand: 32:0 → 403 → 352 → 428 → 354 → 383 → 416 → 437
Route 34, cost: 2220, demand: 33:0 → 330 → 432 → 350 → 371
Route 35, cost: 1019, demand: 31:0 → 394 → 440 → 325 → 388 → 328

Route 36, cost: 2503, demand: 30:0 → 457 → 345 → 379 → 420
Route 37, cost: 645, demand: 19:0 → 257 → 380 → 366
Route 38, cost: 1447, demand: 33:0 → 338 → 271 → 334 → 266 → 374
Route 39, cost: 2101, demand: 33:0 → 284 → 351 → 253 → 460 → 421
total cost =8022.1, Total routes = 40
found in iteration 24891 after 16 seconds

65 km/h, Euclidean distance
Route 0, cost: 1624, demand: 29:0 → 258 → 383 → 416 → 418 → 437
Route 1, cost: 2295, demand: 26:0 → 344 → 274 → 355 → 336
Route 2, cost: 3003, demand: 29:0 → 321 → 278 → 283 → 433 → 287 → 308 → 324
Route 3, cost: 3729, demand: 29:0 → 305 → 307 → 461 → 311 → 367 → 449
Route 4, cost: 1282, demand: 33:0 → 422 → 386 → 348 → 363 → 423 → 309
Route 5, cost: 4675, demand: 33:0 → 403 → 352 → 429 → 402 → 407 → 413 → 342 → 275
Route 6, cost: 2514, demand: 33:0 → 280 → 455 → 404 → 322 → 435
Route 7, cost: 1221, demand: 30:0 → 365 → 417 → 372 → 384 → 300
Route 8, cost: 2356, demand: 32:0 → 394 → 340 → 434 → 450 → 349 → 319 → 255
Route 9, cost: 2635, demand: 33:0 → 447 → 381 → 326 → 290 → 314 → 412
Route 10, cost: 5206, demand: 33:0 → 398 → 310 → 377 → 385 → 456 → 332 → 295 → 293
Route 11, cost: 1850, demand: 33:0 → 452 → 357 → 333 → 391 → 392
Route 12, cost: 1827, demand: 32:0 → 304 → 253 → 425 → 427 → 371
Route 13, cost: 2016, demand: 33:0 → 260 → 338 → 424 → 294 → 292 → 395
Route 14, cost: 1829, demand: 32:0 → 299 → 444 → 400 → 420
Route 15, cost: 2468, demand: 32:0 → 387 → 396 → 369 → 409 → 378 → 261 → 347
Route 16, cost: 2081, demand: 32:0 → 320 → 317 → 428 → 354 → 276 → 346
Route 17, cost: 2573, demand: 31:0 → 256 → 345 → 379 → 373
Route 18, cost: 2108, demand: 33:0 → 312 → 337 → 405 → 362 → 431 → 368
Route 19, cost: 2390, demand: 31:0 → 250 → 448 → 370 → 364
Route 20, cost: 1902, demand: 32:0 → 441 → 335 → 285 → 397 → 426 → 301
Route 21, cost: 1461, demand: 29:0 → 380 → 366 → 453 → 306
Route 22, cost: 1843, demand: 33:0 → 257 → 282 → 272 → 268 → 436 → 323
Route 23, cost: 1976, demand: 33:0 → 318 → 415 → 390 → 284 → 262 → 296
Route 24, cost: 1428, demand: 33:0 → 454 → 356 → 446 → 248 → 286
Route 25, cost: 1852, demand: 31:0 → 279 → 330 → 410 → 419 → 440

```
Route 26, cost: 2454, demand: 33:0 → 359 → 439 → 457 → 254 → 288 → 316 →
358
Route 27, cost: 1261, demand: 33:0 → 273 → 376 → 406 → 269
Route 28, cost: 1781, demand: 33:0 → 281 → 264 → 303 → 460 → 442
Route 29, cost: 1550, demand: 33:0 → 271 → 443 → 331 → 389 → 393
Route 30, cost: 1035, demand: 29:0 → 414 → 408 → 458 → 252 → 328
Route 31, cost: 2124, demand: 32:0 → 351 → 341 → 360 → 401 → 339
Route 32, cost: 2905, demand: 33:0 → 298 → 350 → 259 → 399
Route 33, cost: 1216, demand: 32:0 → 353 → 249 → 451 → 343 → 266 → 374
Route 34, cost: 1907, demand: 32:0 → 388 → 325 → 291 → 302 → 445 → 315 →
297 → 411
Route 35, cost: 1535, demand: 33:0 → 263 → 289 → 430 → 361 → 265 → 334
Route 36, cost: 2504, demand: 32:0 → 270 → 432 → 327 → 375 → 421
Route 37, cost: 1762, demand: 30:0 → 459 → 382 → 329 → 277
Route 38, cost: 440, demand: 24:0 → 313 → 267 → 251 → 438
total cost =8261.8, Total routes = 39
found in iteration 23839 after 14 seconds
```

Appendix E

The CD-ROM

The CD-ROM enclosed with this these contains the source code of the program developed for the thesis, this is found in the directory code.

The datasets from the routeplanning company are on the cd-rom in the directory data. The data for Solomon [4] and Homberger [5] are not on the disk because they are available online.

Included on the CD-ROM are also the spreadsheets containing the data used to calculate the average gap and the max-min-gaps used in chapter 5.

The spreadsheets are:

Pworst.ods: the data for the power function of Worst-removal without recalculating the obtainable reduction.

Pworst_re.ods: the data for the power function of Worst-removal with recalculating the obtainable reduction.

Pshaw.ods: the data for the power function of Shaw-removal

shawParamsTest.ods: the data for the parameters of Shaw-removal

reactionFactor.ods: the data for the reaction factor r used to calculate the scores used to determine the adaptivity of ALNS.

StartingTemparature.ods: Data used to find the percentage of a solution accepted at a possibility of 0.5.

removalpercent.ods: Data used to decide how many customers can be removed at each iteration.

The results of the Homberger instances when the number of iterations was increased to 100.000 and the temperature decrease factor c was set to 0.999975 is included on the CD-ROM. This is appendix F in the file appendix.pdf

The routes created for the four different settings for the real life data day 1, day 3, day 4 and day 5 is included on the CD. This is appendix G in the file appendix.pdf