ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics

Master Thesis
Econometrics and Management Science

An Evolutionary based Decomposition Strategy for the Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows

Author:
Rosita van den Puttelaar
414662

Supervisors:
dr. R. Spliet (EUR)
N. van der Zon MSc (ORTEC)

Co-reader:
dr. K.S. Postek (EUR)

March 17, 2020

Abstract

In this thesis we decompose a large Vehicle Routing Problem with multiple depots, a heterogeneous fleet and customers with time windows. The process, which we call ‘multi-process route optimization’, first splits the VRP over smaller sub-problems and then solves these sub-problems simultaneously using multiple processors. The goal is to obtain solutions of similar quality in substantially less time. We develop an evolutionary based decomposition method with a fitness function that considers both the running time and solution quality. Computational experiments on real world problem instances show that solutions can be obtained in substantially less time. After a few iterations we can even obtain better solutions than without decomposition. After a few iterations of splitting and solving, we alternate our splitting method with the Sweep method to create more diversity in our splits.

Contents

1 Introduction

As more and more customers order online, it becomes more difficult to find the most efficient way to distribute goods. The distribution of goods contributes to the total cost of a product, therefore routes must be constructed such that the total travel distance is minimized and less vehicles are needed. Additionally, customer satisfaction must be increased to compete with competitors. One of the factors that contributes to satisfaction is time. Many retailers offer the option of next day delivery. Therefore, the construction of routes needs to be fast.

The delivery of orders to customers can be seen as a Vehicle Routing Problem (VRP). The classical VRP consists of a set of customers that need to be served with a set of vehicles from a central location, called the depot. The routing problem is to construct vehicle routes such that all customers are served, while minimizing the total transportation costs.

In real life applications, the VRP is often extended with additional characteristics and constraints, like the use of a heterogeneous vehicle fleet. In this thesis, we consider a routing problem for a large retail company. Customers are assigned to a specific time window in which they need to be served. Additionally, there are multiple depots that differ in capacity. Last, we have a heterogeneous vehicle fleet.

Since the VRP is a NP-hard problem (Lenstra and Kan, 1981), real-world problems with large sized instances cannot be solved to optimality within reasonable time. Although it is difficult to obtain an exact solution, approximate solution techniques are available. However, the running time of these techniques can also increase rapidly when the size of the problem increases. Moreover, the huge amount of data that needs to be stored to solve the problem can lead to memory issues.

A possible solution for this is to divide the large VRP over smaller sub-problems and solve these sub-problems simultaneously using multiple processors. We will call this multi-process optimization. In this way we need less memory and the running time decreases. In this thesis we will develop a suitable method to define these sub-problems. The goal is to obtain a quality that is similar to solutions created without splitting in substantially less time.

The research is carried out in cooperation with ORTEC, a large provider of optimization software. Multi-process route optimization is relatively new at ORTEC, and not much research is done on multi-process optimization for routing problems with additional characteristics. Available literature on multi-process optimization also focuses on simple versions of the VRP with one depot, homogeneous fleet and no time windows. There is thus a huge potential for research in the multi-process area.

This thesis is structured as follows. First, we will discuss and review relevant literature in Section 2. In Section 3 we describe the problem, followed by the description of multi-process optimization in Section 4. Next, we discuss our decomposition method in Section 5.1. In Section 6 we discuss some computational experiments to test our method and show the corresponding results. Last, Section 7 contains a conclusion and discussion.

2 Literature review

Researchers have been studying solution methods for the VRP for the past decades. Due to the complexity of real-life problems, the VRP is often extended with additional characteristics and constraints. In this thesis we consider a Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows (MDHVRPTW). We need to take these characteristics into account when we decompose the problem.

Decomposing a problem instance can also be seen as clustering customers and solve the problem for each cluster. Many studies use the term *clustering*. Below we will review some of these studies and point out the differences and similarities with multi-process optimization. The methods can be divided in two main classes, namely cluster-first, route-second heuristics and route-first, cluster-second heuristics (Bowerman et al., 1994). After that we will review some studies that are more similar to multi-process optimization. Finally, we review some cluster techniques that are combined with a Genetic Algorithm.

2.1 Cluster-first, route-second heuristics

Multi-process route optimization has some similarities with the 'cluster-first, route-second' methods known from literature. In this method all customers are clustered and assigned to a vehicle after which routes are computed independently by solving a Traveling Salesman Problem (TSP) (Flood, 1956).

A commonly used method to solve VRPs in this way is the Sweep method (Gillett and Miller, 1974). This method constructs tours by sorting customers according to the polar angle they span with the depot. After the tour is initiated by an arbitrary direction, it is extended by iteratively adding the next customer on the list. If the tour becomes infeasible by an insertion, a new tour is initiated with the chosen customer. Unfortunately, this method gives poor results for the VRP with time windows, because it is difficult to get compatible time windows in a cluster (Labadi et al., 2008).

Distance measures are also often used to cluster customers locations (Anderberg, 1973). In Ho et al. (2008) the Clarke and Wright (1964) saving method is used. The saving value for two customers is equal to the difference between the distance of the customers to the depot and the distance between the two customers. Customers are clustered based on larger saving values. For problems with multiple depots, the authors propose to decompose the problem into a single depot problem first, by assigning customers to the closest depot.

In Ganesh and Narendran (2007) a VRP with delivery and pick-up is considered. In this study an initial number of seed nodes is chosen that will form the clusters. Roughly half of these nodes are the farthest away from the depot and half are nearest. Customers are assigned to the cluster with closest seed node.

Dondo and Cerdá (2007) extend the multi-depot VRP with time windows and heterogeneous fleet. A time-window based heuristic is used to group customers together in clusters, such that the customers can be assigned to one vehicle and the waiting time is minimal. In the next phase clusters are assigned to vehicles, which are then assigned to a depot. The first phase of the algorithm can take some time, because initial routes need to be constructed. Since we want to keep our running time low, we prefer the splitting phase to be fast.

A similar problem is considered in Tansini and Viera (2006). First, an urgency measure is calculated to determine the order in which customers should be assigned to a cluster. This urgency is based on the difference between the proximity to the closest depot and the second closest depot. Proximity is a function of travel distance and distance in time windows to other customers in the cluster. While most studies only use time window to check for compatibility, this study incorporates time windows in the proximity measure.

Koskosidis et al. (1992) decompose the problem by solving a Capacitated Clustering Problem where they approximate the cost of assigning a customer to a vehicle. Since the effect of assigning a customer to a cluster is only known after the routing phase, the algorithm starts with some initial cost matrix. The routes are evaluated after construction and used to update the cost matrix. The algorithm then iterates between the clustering and routing phase, each time updating the cost matrix.

Ostertag (2008) also performs multiple cluster iterations for a multi-depot VRP with time windows. In the first iteration, customers are clustered with a simple heuristics that assigns customers to the closest depot. Because this decomposition method can give an uneven distribution over the sub-problems, sub-problems can be further divided using the Sweep algorithm or can be merged with other sub-problems. The resulting sub-problems are solved with a Memetic Algorithm. In the next iteration new sub-problems are formed by clustering routes based on proximity measures through sweeping and distance. The author compares his method with a fixed decomposition and shows that customers that lie nearly halfway between depots can be better assigned when using multiple iterations resulting in a better objective value.

The *k*-means algorithm can also be used to cluster customer locations. In Geetha et al. (2009) and Kim et al. (2006) the standard algorithm is adjusted by including a priority measure to select locations for a cluster, since the clusters have limited capacity. Both studies set the initial number of clusters equal to the estimated number of vehicles required to serve all customers. The required number of vehicles is estimated by dividing the sum of all demand quantities by the vehicle capacity. After the cluster centroids have been chosen, assignment of customers is done according to a priority rule. The priority rules of the studies are as follows.

Because smaller demands can probably be more easily packed in a cluster, Geetha et al. (2009) assign customers with higher demand first. In Kim et al. (2006) the order of assignment is based on the distance to the centroid of the cluster centroids, called the 'grand' centroid. Customers that are farther away from this grand centroid are assigned first. Customers can only be assigned to a cluster if the vehicle capacity is not exceeded and all customers can still be served by the same vehicle within their time window.

In Reed et al. (2014) the k-means algorithm is used as a preprocessing step. They first use the built-in algorithm in MATLAB without taking the capacity load into account. To adjust for this, they try to move a customer of the cluster with greatest excess demand to a neighbouring cluster. This clustering information is used to perform their ant colony optimization method on each cluster separately.

Barreto et al. (2007) evaluated several cluster analysis based heuristics to group customers. The grouping method that performed best was a hierarchical method that iteratively merges nearest groups. Assigning customers to the nearest depot also gave good results. To determine the closeness of customers, different proximity measures were studied. Proximity measure that gave the best result on average were group average and centroid measures. However, the quality of proximity measure seems instance-dependent.

2.2 Route-first, cluster-second heuristics

The opposite approach, called 'route-first, cluster-second' heuristics, clusters customers in the second phase. Beasley (1983) proposed to relax the vehicle capacity constraint and build one "giant tour" by solving a travelling salesman problem. This tour is then split into a set of individual routes satisfying the capacity constraints. The idea is that in this way customers are clustered based on their closeness.

The giant tour can be constructed in several ways. In Ryan et al. (1993) the order of customers in the giant route are in ascending or descending order of polar angle relative to the depot. Nearest neighbour heuristics are also suited to construct giant tours (Ho et al., 2008).

In the multi-depot heterogeneous VRP of Salhi and Sari (1997) giant tours are constructed per depot. The authors argue that assigning customer to their closest depot can give poor results if customers are nearly halfway between two depots. These *borderline* customer are therefore not assigned beforehand, but inserted after the giant tour is constructed and split. The heterogeneous fleet is handled by assigning the smallest vehicle that can accommodate the total demand of a route.

This basic idea of constructing a giant tour can be extended to deal with additional constraints. Prins et al. (2014) give an overview of associated literature. One extension is to exclude the depot from the tour to increase the flexibility in partitioning. Additionally, this allows for the construction of giant tours in multi-depot problems. For each subsequence of customers the best depot and its optimal insertion can be determined (Vidal et al., 2014). Heterogeneous fleet can be handled in the same way. Labadi et al. (2008) incorporate time windows in the splitting procedure by doing a feasibility check on the time windows.

Several studies alternate between the two solution spaces (giant tours and routing solutions). This approach can also be combined with genetic algorithms or ant colony optimization. In the former, giant tours are encoded as chromosomes and a crossover operator is used to generate new chromosomes (Lacomme et al., 2001; Prins, 2004). In Santos et al. (2010) giant tours are built by ants based on pheromone information.

2.3 Decomposition strategies

In this thesis we develop a more general decomposition strategy. The main difference between the mentioned cluster heuristics and multi-process optimization is that the main goal of multi-process optimization is to reduce the running time. Another difference is that in multi-process optimization customers are assigned to a split that can consists of multiple vehicles. Furthermore, multiple iterations can be performed to determine the optimal split. We can nevertheless use these heuristics as starting point and change the capacity condition of a cluster.

The term multi-process optimization is not used in the literature. However, some researchers have been using similar methods under a different name. Below we review some of these studies.

Taillard (1993) uses the Sweep method to divide a problem instance into a pre-specified number of polar regions. Approximately the same number of customers are assigned to each region. These regions can be further divided based on the radius of the locations. Each sub-problem is treated as an independent VRP and can be solved in parallel. This method works well for problems with customers uniformly divided around the depot, as shown by high quality solutions for classical problems. A relatively simple VRP is considered in this study and the proposed method becomes difficult when the problem has multiple depots. The method can, however, be applied if we first assign customers to a depot.

Another example of a decomposition strategy is the D-Ants approach of Reimann et al. (2004). The problem is decomposed by applying the Sweep method to the centres of gravity of an initial route solution. The authors show that decomposing the problem and then solving these smaller sub-problems improves the effectiveness of the solving strategy. An advantage is that this method can be applied to any vehicle routing problem, including VRPs with time windows and multiple depots. The authors also suggest to decompose the problem in several single-depot problems first, and then apply their algorithm.

Taillard and Voss (2002) introduce a framework that tries to standardize the decomposition procedure. The method is named as *POPMUSIC*, which stands for Partial OPtimization Metaheuristic Under Special Intensification Conditions. An initial solution is split into p parts using some relatedness measure. A sub-problem is then created by aggregating the closest r parts around a chosen seed part. The sub-problems can be re-solved. If solving the sub-problem yields an improvement, the complete solution is updated.

2.4 Cluster-based Genetic Algorithms

Cluster techniques can also be combined with a Genetic Algorithm (GA) to iteratively improve the sub-problem distribution (Cheng and Wang, 2009). Researcher Thangiah was one of the first to use GA for the clustering phase in the 'cluster-first, route-second' methodology. In Thangiah (1993) a system is developed to solve the VRPTW. Seed angles originating from the depot are used to partition customers in clusters, one for each vehicle. The seed angles are computed using a fixed angle and an offset from the fixed angle. The GA is used to search for the set of offset that minimizes the total operating cost. To evaluate the clusters produced by this algorithm, a route is formed with a least-cost insertion heuristic, where time window and capacity violations are penalized. The crossover operator exchanges some randomly selected offsets.

Often, chromosomes directly represent the cluster distributions. In Cheng and Wang (2009) chromosomes denote by which vehicle the customers are served. New chromosomes are created by recombining two chromosomes with a two-point crossover. A mutation operator is applied with probability p_m and randomly swaps the vehicle of two selected customers. The fitness value of a chromosome is obtained by summing the objective values of the corresponding sub-problems. A drawback of this method is that in each iteration all sub-problems need to be solved to obtain the fitness value, since route information is not saved. This is inefficient since not all clusters have been changed after crossover. Furthermore, if the cluster size increases, solving all clusters is more time consuming.

Another example of a cluster-based GA is that of Pankratz (2005), where each gene of a chromosome corresponds to a cluster of customers served by the same vehicle. Offspring is produced by inserting one or more clusters from the second parent into the first parent. Duplicate vehicles and customers that originally belong to the first parent are removed from the solution. Customers that become unassigned by this removal are re-inserted. Finally, a mutation operator eliminates a cluster and re-inserts the associated customers with an insertion heuristic. The fitness values of the initial population are obtained by creating a route for each vehicle with an insertion heuristic. Routes are re-adjusted after applying genetic operators by deleting customers that were part of the regrouping and reinsert them.

In Maulik and Bandyopadhyay (2000) an evolutionary based variant of the k-means algorithm is developed where GA is used to determine the cluster centroids. Chromosomes represent the cluster centroids and the fitness value of each chromosome is calculated by the sum of distances from customers to the centroid. Single-point crossover is used to create offspring where points to the right of some integer are exchanged between parents. Last, new solutions are mutated with a fixed probability by multiplying the centroid point by some $\delta \in [0, 1]$.

Several other studies use GA to determine cluster distribution (Potter and Bossomaier, 1995; Thangiah and Salhi, 2001; Jorgensen et al., 2007), and in almost all studies evaluating chromosomes involves solving the routing problem. If the problem size increases, this can be very time consuming. We will therefore try to develop a GA where the fitness evaluation will be fast.

3 Problem description

In this section we describe the problem of the retail company. A more generic VRP in which vehicles have limited capacity is called the Capacitated Vehicle Routing Problem (CVRP). We first give a description of the CVRP before we discuss the specific characteristics of our problem.

3.1 Capacitated vehicle routing problem

The delivery of orders to customers can be viewed as a CVRP. The road network is defined on a directed graph $G = (V, A)$, where V denotes the set of vertices and A the set of arcs. The set A contains arcs between each pair of nodes. The set of vertices can be divided into a set of customer nodes, N, and a depot node, $\hat{d}$. Each customer has a certain demand quantity. The available vehicles form the set K and have a certain capacity Q. Orders are assigned to a vehicle in such a way that the total volume of the orders does not exceed the vehicle capacity. A *route* is defined by a sequence of customers assigned to one vehicle.

The goal is to find a set of routes for the vehicle fleet to serve the set of customers such that the transportation costs of the routes is as low as possible.

3.2 Components

In this subsection we explain in more detail the three components of our extended CVRP, namely the customers, depots and vehicles. For each component several characteristics and constraints are discussed.

3.2.1 Customers

Each customer location is represented by a node in the graph. The demand of customer $i \in N$ is measured by two units, denoted by $P_i = (P_{1,i}, P_{2,i})$. Orders have to be delivered within a given time window. That is, for each customer $i \in N$ there is a time window $[a_i, b_i]$. This means that orders can only be delivered after a_i and before b_i. If a vehicle arrives at the customer location before a_i it has to wait. Last, each customer location has a fixed service duration, s_i, that represents the time needed to handle the order.

3.2.2 Depots

Instead of having one depot, the considered retail company operates from multiple depots, denoted by the set D. There are two different types of depots: main depots and sub-depots. For the main depot we can assume that the capacity is not restricting. There are, however, a limited number of vehicles available. At the sub-depots both the number of vehicles and the capacity are restricting. Since some vehicles are allowed to reload at the depot, the capacity of the depots can be larger than the sum of vehicle capacities. Customers can be supplied from both types of depots.

3.2.3 Vehicles

At each depot $d \in D$, a set of vehicles, K_d, is available. The fleet set consists of heterogeneous vehicles, with the different types denoted by the set W. Each type $w \in W$ has a capacity $Q_w = (Q_{1,w}, Q_{2,w})$, driving speed s_w, and operating cost o_w. The fleet set thus consists of heterogeneous vehicles, with different capacities, driving speed and operating costs.

Vehicles can have a route assigned to it. A route starts with loading at a depot and for each route it is pre-specified at which depot it must start. After visiting the assigned customers, the vehicle returns to the same depot. Some vehicles can load again and supply additional customers. Each time the vehicle reloads at the depot, we say that it starts a new *trip*. A route can thus consist of multiple trips. For each vehicle it is specified how many trips it can execute.

Each vehicle has an earliest start time and a latest finish time. Vehicles can only operate between these two times. There are also break requirements that need to be taken into account. Vehicles have to take a break of x minutes for a working time between y_1 and y_2 minutes. Last, each vehicle has a maximum allowed working time.

4 Multi-process route optimization

We will now explain the different stages of multi-process route optimization. We first decompose the problem after which we will solve each sub-problem individually using a specified solving procedure. Below these stages are described in more detail.

4.1 Problem decomposition

The first step in multi-process optimization is to decompose the problem in several sub-problems. The idea of problem decomposition is to decompose a large VRP over smaller sub-problems and solve each of these simultaneously using multiple processors. With problem decomposition we mean that the orders, depots and vehicles are divided over the different sub-problems. Orders and vehicles can only occur in one sub-problem, while depots can be in more sub-problems. Aggregating the sub-problems constitutes a complete solution.

The main objective of multi-process optimization is to obtain a solution quality that is similar to solutions created without splitting in substantially less time. The quality of a solution is denoted by a cost function. A lower cost yields a better solution. We only consider a solution to be feasible if all customers are served.

Our algorithm consists of different types of iterations to determine the best decomposition. Figure 1 gives a schematic overview of the procedure. A multi-process iteration is defined as the process starting with decomposition and ending with consolidation. During the decomposition phase, sub-problem decompositions are initialized and improved using a genetic algorithm, which we will explain in Section 5. The best decomposition is returned and its sub-problems are solved. After consolidation, we can start a new multi-process iteration or return the consolidated solution.

In the first multi-process iteration we assume that all customers are unplanned and we therefore split on the customer-level. From the second iteration onward, when routes are constructed by solving sub-problems, the splitting is done on the route-level which keeps important information of the solution. Due to the current implementation of multi-process optimization in the ORTEC software, we keep the customers and vehicles together during the algorithm. We consider customers in a route as planned and all remaining customers as unplanned.

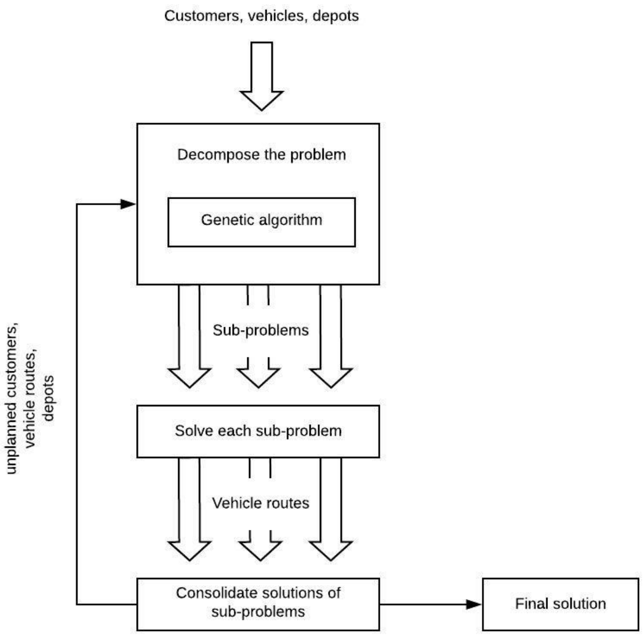

Customers, vehicles, depots
Decompose the problem
Genetic algorithm
Sub-problems
Solve each sub-problem
Vehicle routes
Consolidate solutions of sub-problems
Final solution
unplanned customers, vehicle routes, depots

Figure 1: Schematic overview of multi-process route optimization

We point out that we mainly focus on the splitting phase, not on solving the sub-problems. We do however investigate if a good decomposition depends on how the sub-problems are solved. In the next sections we will explain how we solve the sub-problems and how we relate this to our decomposition method.

4.2 Solving sub-problems

After we decomposed the problem, the next step is to solve the sub-problems independently (and simultaneously). The solving is done with an optimization system from ORTEC, called CVRS. This system contains many heuristics and metaheuristics to solve vehicle routing problems. The solving process can be divided in different parts. First a constructive heuristic is used to create a feasible solution. After finding a solution, local search is applied to improve the solution. Optionally, a ruin-and-recreate principle is performed where a part of the solution in removed and constructed again to escape local minima.

The product is highly configurable. A command template is used to specify the combination of algorithms that should be used and in which order. There exists a template that is specifically created to fit the needs of the client. This template gives high quality solutions, but at the expense of a high running time. To keep our method more general, we use a basic template that is not designed to fit specific needs. A good decomposition probably depends on which solving method is used for the sub-problems. We therefore will consider two different templates. We will investigate how we should define some elements of the genetic algorithm if the solution method changes.

The two different templates we consider have the following characteristics. The first template produces a high quality solution at the cost of a longer running time, while the second template solves the problem much faster but also has a lower quality solution. Both templates have a construction phase followed by a local search procedure. The slow template additionally performs the ruin-and-recreate principle. The command template for the second multi-process iteration also has a constructions phase, but only to insert any unplanned customers.

After the sub-problems are solved, the solutions are consolidated. A new multi-process iteration can start, splitting the problem on the route-level.

5 Genetic algorithm for decomposition

In this thesis we use a genetic algorithm for our decomposition method. We will perform multiple multi-process iterations, therefore our algorithm needs to be defined for both the customer-level as well as the route-level. We first describe the main idea of a genetic algorithm before we define the core concepts of the algorithm.

5.1 Outline genetic algorithm

The genetic algorithm is a search heuristic that is inspired by the theory on natural evolution. The fittest individuals of the population are selected to produce offspring. In Algorithm 1 a description of our genetic algorithm is given. Our genetic algorithm starts with forming an initial population with sub-problem decompositions generated by a fast splitting procedure. For each individual in the starting population the fitness value is calculated. Since solving all sub-problems is time-consuming, our fitness function is an approximation of the costs. We will explain in Section 5.2 how this approximation is defined.

After the initial population is formed the following steps will be repeated until a stopping criterion is met. First, two parents are selected from the population and are combined through a crossover operator to generate two offspring. To maintain genetic diversity, different mutation operators are used on the offspring. The fittest individual will enter the population. We keep a fixed number of individuals in our population, therefore one of the z worst individuals is removed. If the stopping criterion is met, we evaluate the chromosome with the best fitness value.

Algorithm 1 Overview genetic algorithm

```
1: create an initial population with $n$ individuals
2: for $iteration = 1$ to $r$ do
3:     select two parents $P_1$ and $P_2$
4:     apply the crossover operator to form offspring $C_1$ and $C_2$
5:     for each mutation operator $m$ do
6:         if $random < p_m$ then
7:             apply mutation operator $m$
8:     add best offspring to population and
9:     randomly remove one of the $z$ worst individuals
10: return individual with the best fitness value
```

5.2 Chromosomes and evaluation

An important setting in the genetic algorithm is the encoding of solutions into chromosomes. Each chromosome in our population represents a sub-problem distribution. For each customer, vehicle, and depot it is denoted to which sub-problem it belongs. For simplicity, we assume that the depot capacity is defined by the available vehicle capacity. Hence we do not need to divide the depot capacity over sub-problems if the depot occurs in more than one sub-problem. The difference between the depot capacity and total vehicle capacity is small in our considered instances. The number of sub-problems is variable and can vary over different chromosomes. There are, however, a limited number of processors available giving us an upper limit for the number of sub-problems.

Since it is very expensive with respect to running time to solve each sub-problem, we approximate the quality of a sub-problem distribution without solving it. We focus on both the running time and the solution quality. We therefore split our algorithm in two phases. In each phase we use a different fitness function to represent the main focus of that phase.

In the first phase of our genetic algorithm we focus more on the running time. With the fitness function in this phase we aim to get the desired number of sub-problems as well as the desired size. We will refer to this fitness function as the *size* fitness function. Although we focus more on the sub-problem size, we also add an indication of the solution quality and feasibility.

In the second part we focus more on the the solution quality using a different fitness function, called the *approximate cost* fitness function. We only consider the best m chromosomes of the first phase and re-calculate the fitness function. From here, we continue our algorithm as before.

5.2.1 Sub-problem size

With the size fitness function we focus on the running time. This is done by focusing on the number of sub-problems and their sizes, as the running time decreases if a sub-problem is smaller. As mentioned before, we also add an indication of the solution quality and feasibility to avoid low solution quality in the end. Normally, a higher fitness function constitutes a better solution, but in our fitness function, good decompositions are featured with a lower fitness value.

The size fitness function consists of multiple parts, each corresponding to a different quality measure. The importance of each measure is determined by a parameter. The fitness function of a chromosome S looks as follows:

$$F(S) = \alpha G(S) + \beta H(S) + \gamma I(S) + \delta K(S) + \zeta L(S). \tag{1}$$

In the rest of this section we will discuss these five quality measures.

Within sub-problem proximity
Since the operating costs depend on the driving time and distance, customers should preferably be close to each other. The first quality measure, $G(S)$, therefore calculates the within sub-problem proximity. We calculate this measure as follows: Let $S = \{S_1, S_2, ..., S_R\}$ be the set of R sub-problems and N_r the set of customers in sub-problem r. The centre of gravity z_r of sub-problem r is denoted by:

$$z_r = \left(\frac{\sum_{i \in N_r} x_i}{|N_r|}, \frac{\sum_{i \in N_r} y_i}{|N_r|} \right) \tag{2}$$

The within sub-problem proximity is defined as the average distance between customers and the centre of gravity of their sub-problem. It is denoted as follows:

$$G(S) = \sum_{r=1}^{R} \frac{1}{|N_r|} \sum_{i \in N_r} \left\| (x_i, y_i) - z_r \right\|, \tag{3}$$

where S represents a chromosome with a sub-problem distribution and z_r the centre of gravity of sub-problem S_r.

Feasible capacity
With $H(S)$ we check for feasibility of the vehicle capacity. If the total vehicle capacity is lower than the order quantity, we add this capacity shortage to the fitness value. Infeasible decompositions are thus featured with a higher fitness value. Since infeasible solutions can have promising properties and unplanned orders are considered in next iterations, we do not delete them.

Let D_r and K_r denote the set of depots and vehicles in sub-problem r, respectively. For simplicity we denote the customer demand by P and the vehicle capacity by Q. The total order quantity and total vehicle capacity of sub-problem r are then denoted by:

$$TP_r = \sum_{i \in N_r} P_i, \quad TQ_r = \sum_{j \in K_r} Q_j. \tag{4}$$

The feasibility check can then be defined as follows:

$$H_1(S) = \sum_{r=1}^{R} I_{TP_r \geq TQ_r} \times (TP_r - TQ_r), \tag{5}$$

were I is an indicator function that represents whether there is a capacity shortage.

Besides checking for feasibility, we also want an equal distribution of the remaining capacity over the sub-problems. With remaining capacity we mean the difference between the total vehicle capacity and the total order quantity, assuming we can fully exploit the vehicle capacity. It might be beneficial to have some spare capacity, since additional constraint make it harder to use the full capacity of vehicles. The difference between the maximum and minimum remaining capacity is added to the fitness value, denoted as follows:

$$H_2(S) = \max_{r \in R}(TQ_r - TP_r) - \min_{r \in R}(TQ_r - TP_r). \tag{6}$$

The total value of this quality measure is defined as the sum of H_1 and H_2.

Depot assignment
Since it is probably cheapest to supply a customer from its closest depot, we add a penalty $I(S)$ to the fitness function if the closest depot of a customers is not included in the same sub-problem. This penalty is equal to the difference in distance between the customer and its closest depot and

the customer and its closest depot in the sub-problem:

$$I(S) = \sum_{r=1}^{R} \sum_{i \in N_r} \left(\min_{d \in D_r} \|i - d\| - \min_{d \in D} \|i - d\| \right). \tag{7}$$

Sub-problem size
Since all sub-problems are solved in parallel, the running time is defined by the slowest sub-problem. If we use a slow solving template, the difference in running time can be substantial if there is a big difference in problem size. If we use a fast template on the other hand, it might not matter much. To reduce the difference in running time between sub-problems we can force an equal distribution of the customers and vehicles over the sub-problems. Therefore, $K(S)$ is defined as the standard deviation of the number of vehicles and customers per sub-problem.

Besides an equal distribution, we can also reduce the running time by increasing the number of sub-problems. By setting a minimum and maximum number of sub-problems, we penalise decompositions with $L(S)$ if the number of sub-problems is not within the desired values. The value of $L(S)$ is equal to the difference between the number of sub-problems and the maximum or minimum, depending on which is violated.

5.2.2 Approximate routing cost

In the second part of our genetic algorithm we use a fitness function that can better predict the plan costs. Following the approach of Nicola et al. (2019), we consider a large set of potential variables and with the help of a linear regression we identify the relation between these variables and the total cost. All variables are measured per sub-problem. The variable value for a chromosome is obtained by summing the values of the corresponding sub-problems. We divided the variables in four categories, which are explained below.

Distance related measures
Since the total cost depends on the travelled distance between customers, we include several distance related measures in our model. These include the sum (SumP), variance (VarP), minimum (MinP) and maximum (MaxP) of distances between all pair of nodes. Furthermore, we include the sum of distances to the nearest (SumMinP) and the farthest (SumMaxP) neighbour of each customer. Due to time windows, some customers can not be visited after each other. We therefore only consider links between customers that satisfy the time window constraints.

Like the within sub-problem proximity, we also consider distances to the centre. We include the following variables. The minimum (MinM), the maximum (MaxM), the sum (SumM) and the variance (VarM) of distances between all customers and the sub-problem centre. Last, we consider the distances between customers and their closest depot in the sub-problem, measured by the variables SumD and VarD, denoting the sum and variance of the mentioned distance, respectively.

Time window related measures
As mentioned before, time windows can have an effect on the cost. We measure the influence of time windows with the variance (VarTW) of the length of the customer time windows. We also consider the sum (SumTW) of the time window lengths. Note that this value is the same for each chromosome, we therefore only consider this variable if we compare different instances. The amount of overlap in time window can also influence the routing possibilities. The variables SumOverlap, AvgOverlap and VarOverlap measure the sum, average and variance of time window overlap between customers, respectively.

Capacity related measures
The costs increase if more vehicles are used. The number of vehicles necessary is dependent on both the vehicle capacity and the customer quantities. These effects are captured with the following variables. First we add the total-demand/average-vehicle-capacity (TotD/AvCap) ratio, which estimates the minimum number of vehicles necessary to supply all customers. The average-vehicle-capacity/average-demand (AvCap/AvD) ratio represents the average number of customers that can be supplied by one vehicle. If a vehicle can perform multiple trips, we multiply its capacity by the number of allowed trips.

Additional measures
Besides taking the constraints into account, we also include some general measures. The first is the product of variances of the x and y coordinates of the customer locations (VarX $\times$ VarY). If customers are scattered in both directions, the travel distance increases. Last, we include the number of customers, n, as a variable in our model. This variable is only used if we compare different instances, since the number of customers is the same for each chromosome within an instance.

5.3 Population management

In this section we determine which chromosomes may enter the population and which chromosomes are removed from the population.

We will keep a fixed number n of individuals in our population. Individuals in our population represent different problem decompositions. To initialize the population we will use a relatively simple and fast splitting strategy. We will explain our initialization method in Section 5.4.

After we generated the initial population, the algorithm performs r_1 iterations with the size fitness function and r_2 with the approximate cost function. In each iteration offspring is generated from the selected parents, from which only the fittest one can enter the population. We only allow the newly generated individual to enter the population if the fitness value is lower than the worst individual in the population.

Since the population size is fixed, the new individual has to replace an existing individual. Since the chromosome with the worst fitness value does not necessarily result in the highest cost, we randomly replace one of the z worst individuals. By removing chromosomes with a high fitness value, the average quality of the population should increase

5.4 Initial population

In this section we explain how we constructed the initial population by creating different sub-problem decompositions. In the first multi-process iteration, we form the initial population on the customer-level. After that, routes are constructed and the population is formed on the route-level.

5.4.1 Customer-level

In the first iteration we form our initial population on the customer level in the following way. We first choose the number of sub-problems k. Since we do not know the optimal number of sub-problems in advance we vary this number by drawing a random number between k_{min} and k_{max}. Next, we will select k seed customers as follows. The first seed customer is chosen randomly, while the next customers are iteratively selected based on their distance to the closest seed customer(s) selected so far. The probability of choosing a customer as seed is proportional to its distance from the closest seed.

This initialization takes extra time compared to randomly selecting customers, but in this way the initial sub-problems are more spread out.

Before we form the sub-problems by assigning all remaining customers to the closest seed customer, we first determine the maximum number of customers per sub-problems. Since we want our sub-problems to be about equal size, we only allow a maximum number of customers per sub-problem. As starting point we use the number of customers divided by the number of sub-problems. Since the sub-problems do not necessarily have to be equal in size, we add some "slack" to this maximum number of customers. This new maximum will be used when assigning the customers. The slack can be adjusted manually.

The order in which customers are assigned is based on a regret value. This value measures the "regret" of not assigning the customer to its closest seed and is equal to the difference in distance to the closest seed and the second closest seed. Customers with a higher regret are assigned first. If the maximum number of customers for a sub-problem is reached, customers are assigned to their second closest seed (if possible).

We continue with forming the sub-problems by adding the depot that is closest to the centre of each sub-problem. If a depot is only added to one sub-problem, all corresponding vehicles are added to this sub-problem. If a depot occurs in more than one sub-problem, we distribute the vehicles such that the number of vehicles is proportional to the number of customers. It can also happen that a depot does not occur in any sub-problem. We make a distinction here between the main depot and sub-depots. Vehicles of the main depot are more likely to travel long distances, because the sub-depots can most likely not supply all customers nearby due to their low capacity.

Therefore, the main depot is added to all sub-problems and the vehicles are divided proportional to the number of customers. Sub-depots are more likely to supply customers nearby. We therefore select the sub-problem with closest centre and add the depot with corresponding vehicles to it.

It is desirable that our initial population contains feasible decompositions, since this is the starting point of our algorithm. We therefore check the capacity restriction. If there is a capacity shortage, random vehicles of the main depot are re-assigned to this sub-problem, provided that the switch does not result in a capacity shortage of the giving sub-problem. Note that, due to other restrictions, unplanned orders are still possible after solving.

We repeat this procedure until our population has the desired size.

5.4.2 Route level

From the second multi-process iteration onward, we form our population on a route-level, that is, customers in the same route will always be in the same sub-problem. We start our initial population in a similar way as in the first iteration, by choosing k random seed routes and assigning each remaining route to the closest seed route. Distance are calculated between the centre of gravity of a seed route and each customer in other routes (Ostertag, 2008). The centre of gravity of a route is calculated in the same way as for the centre of gravity of a sub-problem. We again add a capacity limit on each sub-problem and assign routes according to largest regret value.

If the solution contains unplanned customers, we assign these customers to the closest seed. Empty vehicles are divided over sub-problems in which its depot occurs. The division is proportional to the number of customers per sub-problem.

5.4.3 Assign to closest depot

The sub-problems are disjunct and solved independently, so there is no exchange of information about feasibility possible. The depot capacity might therefore be exceeded if it occurs in more sub-problems. For simplicity we assumed that the depot capacity is equal to the capacity of the available vehicles, but if we do not want to make this assumption, a simple solution is to split the problem such that a depot only occurs in one sub-problem. To make sure the decompositions are feasible with respect to capacity, we also add the main depot to sub-problems that have a capacity shortage. The capacity of the main depot is not restricting, however, we do need to divide the vehicles of the main depot over the sub-problems. We randomly add vehicles until there is enough capacity.

5.5 Selection and crossover operators

To form new decompositions, we first need to select two parent chromosomes from the population through a selection procedure. We use a binary tournament selection where we randomly select two chromosomes S_1 and S_2 from the population. The fittest of the two individuals is selected as the first parent. We repeat this selection procedure for the second parent. We draw with replacement, hence parent 2 can be equal to parent 1.

These parents are recombined through a crossover-operator to create offspring that hopefully inherit good properties of the parent solutions. Recombination is sub-problem-oriented, that is, the operator works on sub-problems rather than individual objects such as customers. The used operator is as follows. Given two parents, we first select a random sub-problem from the first parent and add this sub-problem to the second parent. As a result of this operation some customers and vehicles occur twice. To repair the new decomposition, we remove these duplicate customers and vehicles that originally belong to the second parent. The second offspring is generated by repeating the above procedure and reversing the roles of the parents.

Applying the described crossover operator can also result in sub-problems without customers and/or trips. If this happens, we repair the decomposition by removing it. Unassigned orders by this removal are assigned to the sub-problem with the closest centre and vehicles are proportionally distributed over the sub-problems that contain their depot.

We use the same selection procedure and crossover operator for the customer-level and the route-level.

5.6 Mutation operators

Mutation operators are used to diversify the search directions and avoid early convergence. Each stage of the algorithm may require different operators. We therefore have multiple dynamic mutation operators. Each mutation operator m has its own mutation probability p_m and forms offspring. The mutation ratio is updated each iteration according to the evaluation results from the offspring it produces by multiplying the probability with the relative improvement. As a result, good mutation operators have a higher probability of being selected.

5.6.1 Sub-problem level

We split our mutation operators in two groups, based on how they mutate the chromosome. The first class of mutation operators we consider works on the sub-problem level, to have high diversity in sub-problem constitutions. This means that all customers and vehicles of the selected sub-problem are affected by the change. We consider the following operators: removing a sub-problem, splitting a sub-problem and a combination of merging and splitting sub-problems.

Remove sub-problem
The first mutation operator removes a random sub-problem and re-inserts the customers and vehicles in another sub-problem. To select a sub-problem we calculate the fitness value per sub-problem. Bad sub-problems are featured with a higher fitness value and therefore have a higher probability of being selected. Since the fitness value of a single sub-problem also depends on the number of customers in a sub-problem, we look at the average fitness value per customer. The probability of being selected is proportional to the average fitness value per customer of a sub-problem.

The re-assignment happens as follows. Unplanned customers from the removed sub-problem are assigned to the sub-problem with closest centre of gravity. Planned customers in a route (with the corresponding vehicle) are all assigned to the sub-problem that is closest to the route centre. Empty vehicles are divided over sub-problems in which their depot occurs. The division of vehicles is proportional to the number of customers in the sub-problems. If a depot in the removed sub-problem does not occur in any sub-problem, we add it to the sub-problem with the closest centre of gravity. All corresponding vehicles are also added to this sub-problem.

Split sub-problem
To create more sub-problems, this mutation operator splits a selected sub-problem. Here, sub-problems with more customers have a higher probability of being selected. We select the two customers in the selected sub-

problem that are furthest away from each other. All remaining customers are assigned to the closest customer to form two sub-problems.

If the sub-problem contains one depot we divide the corresponding vehicles over the sub-problems proportional to the number of customers. If the sub-problem contains more than one depot, we try to assign the closest depot to each sub-problem. If both sub-problems have the same closest depot, we divide the vehicles proportional to the number of orders. Otherwise we assign all vehicles of the corresponding depot to the sub-problem. The vehicles of the main depot are always divided proportionally over the two sub-problems.

Merge and split sub-problem
The last mutation operator keeps the number of sub-problems the same. First, the two sub-problems with closest centres are merged together. This merged sub-problem is split with the described splitting method above.

5.6.2 Customer/vehicle level

The next mutation operator only affects some vehicles or customers in a sub-problem. Only a few customers or vehicles change sub-problems.

Re-assign customers and vehicles
In the first multi-process iteration all customers are unplanned. The mutation operator selects x customers an re-assigns them to a different sub-problem. Customers that are further away from their sub-problem centre and closer to another sub-problem have a higher probability of being selected. This probability is proportional to the distance to the centre minus the distance to another sub-problem. Distance of a customer to a sub-problem is defined as the minimum distance between the selected customer and all customers in the sub-problem. The selected customers are re-assigned to the closest sub-problem.

From the second multi-process iteration, customers are assigned to vehicles or are still unplanned. The mutation operator re-assigns a vehicle together with the planned customers to a different sub-problem. Vehicles with their route further away from the sub-problem centre and close to another sub-problem have a higher probability of being selected by this mutation operator. Distances are calculated between the route centre of gravity and single customers in other sub-problems. The probability of selecting an unplanned customer is defined in the same way as in the first multi-process iteration. Both routes and unplanned customers can be selected in this iteration, based on their probability.

Additionally to non-empty vehicles, we also re-assign empty vehicles to divide the capacity more equally over the sub-problems. Sub-problems with more vehicles in proportion to customers are thus more likely selected to give vehicles and sub-problems with few vehicles are more likely to receive vehicles. The probabilities are proportional to the vehicle/customer ratio and the customer/vehicle ratio, respectively. The number of vehicles being exchanged is y, or until the difference in vehicle/customer ratio between the sub-problem is smaller than 5. The mutation operator works the same in both multi-process iterations.

6 Computational results

The computational experiments are done on a practical and on a theoretical case. The instances of the practical case consist of several real-world instances of one of ORTEC's clients. The instances of the theoretical case are based on the Solomon benchmark instances. We used the original instances to create instances with 1000, 1500 and 2000 customers.

6.1 Algorithm configuration

Our genetic algorithm is implemented in the C++ programming language. All results are obtained using a laptop with Windows 10 with a i7-8650U processor (2.11 GHz) using 8 cores and 16 GB RAM. Note that, since we only have 8 cores, we can only solve 8 sub-problems at the time. If chromosomes contain more than 8 sub-problems, we still consider the maximum duration of all sub-problems.

Next, the following parameters need to be configured in our algorithm. We created an initial population of $n = 25$ individuals. Since we aim for a fast splitting procedure, we also want our initialization to be fast. The duration of the initialization is partly defined by the number of individuals. $n = 25$ gives a good balance between running time and sufficient variation. For each individual in our initial population we draw a random number between 5 and 8 that defines the number of sub-problems k. We chose 5 and 8, since the maximum and minimum allowed sub-problems are around these numbers. With the mutation/crossover operators, the algorithm still has room to create more or less sub-problems. The 'slack' on the maximum number of customers per sub-problems is set to 50, which is around 2.5% for most of our considered instances.

After that, our genetic algorithm performs $r_1 = 75$ iterations with the size fitness function and $r_2 = 25$ iterations with the approximate cost function. The number of iterations can be increased, but at a cost of a higher running time. We therefore choose to keep the number of iterations low. We need more iterations with the size fitness function to ensure feasibility. We assume that the quality of the chromosomes is reasonable after the first phase, therefore only 25 iterations with the approximate cost fitness function are performed. Only the $m = 20$ best individuals are re-evaluated and used for the second part, to filter out possible bad solutions. The fittest individual of each iteration will replace one of the $z = 5$ worst individuals.

The starting values of the mutation operators are 0.4, and the maximum is 1.0. To avoid low mutation rates and as a consequence low diversity, we set a minimum of 0.1 for each operator. The number of customers that is re-assigned by the mutation operator is $x = 50$ and the number of empty vehicles is $y = 50$.

6.2 Practical case

The instances of the practical case are real-world problem instances provided by one of ORTEC's clients, a retail company.

6.2.1 Problem instances

The problem instances are provided by the considered retail company. The instances are based on four distinct service areas, denoted by A, B, C, and D. Each service area provides five instance consisting of customer demand for different days and/or day parts (morning and afternoon). This gives a total of 20 instances. A description per instance can be found in Table 10 in Appendix B. In Table 1 we give a summary of the instances per region. The denoted values are the average of each region.

Table 1: Average value of the number of depots, vehicles and customers per region. The different vehicle types are numbered by Vehicle1, Vehicle2, etc.

Region	#Customers	#Depots	#Vehicles1	#Vehicles2	#Vehicles3	#Vehicles4	#Vehicles5
A	2048	7	0	11	112	150	18
B	2721	3	0	13	165	150	0
C	2244	6	0	14	166	150	12
D	2053	4	3	22	127	150	0

Figure 2 shows the distribution of customers and depots for the four different regions. The depots are denoted by the black squares. From these figures we can see that the distribution is quite different per region. In the rest of this section we will discuss how this influences our estimates.

For each customer demand the time window, service time and order quantities are specified. Each service area has its own set of depots and vehicles. The set of depots always consists of one main depot with infinite capacity and several sub-depots with finite capacity. The available depots and vehicles per day part can vary in the same area.

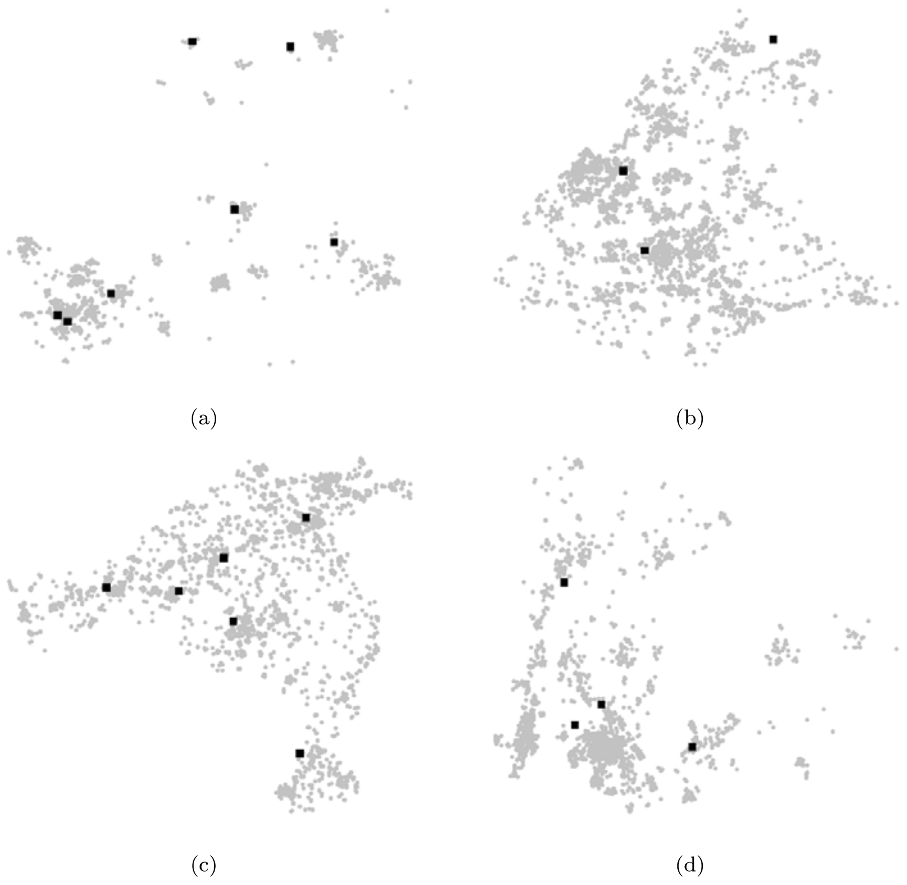


Figure 2: Distribution of customers for instances A, B, C, and D_1. Depot locations are indicated by the black squares.

The retail company has five different types of vehicles available, having different driving speed, operating costs, and capacity. The capacity of a vehicle is denoted by two units. Additionally, vehicles have break requirements. Each route with a working time of at least 75 minutes must contain a break of 15 minutes. If a route has a duration of at least 225 minutes, it needs another break of 15 minutes. Waiting at customer locations is also considered as a break, provided that the waiting time is at least 15 minutes. Moreover, the vehicles have a maximum duration of 8 hours. An overview of the different vehicle types and their operating cost can be found in Table 2. In Table 11 in Appendix B we give an overview of the division of vehicle types over the depots.

6.2.2 Tuning of the fitness function

Our decomposition method strongly depends on a good fitness approximation, which consists of different quality measures, each multiplied by a parameter. Before we evaluate the quality of the fitness function, we first

Table 2: Overview of the different vehicle types.

Type	Speed factor	Cost perHour	Cost perDistanceUnit	Cost perRoute	Cost perOrder	Capacity unit1	Capacity unit2
1	0.47	30	12	0	0	180	250
2	0.85	60	8	0	10	600	875
3	1.00	200	0.1	2000	20	600	950
4	1.00	200	0.1	4000	20	600	950
5	1.15	200	0.1	2000	20	600	950

explain how we have chosen the parameter values. To tune the parameters of our algorithm we use different test instances. To see the effect of how the customer locations are distributed, we select one instance of each region for our test cases. We also make sure that there is some variation in the number of depots and the number of customers in these instances. In Table 3 we give an overview of the used test instances.

Table 3: Description of the instances used for determining the parameters of the fitness functions.

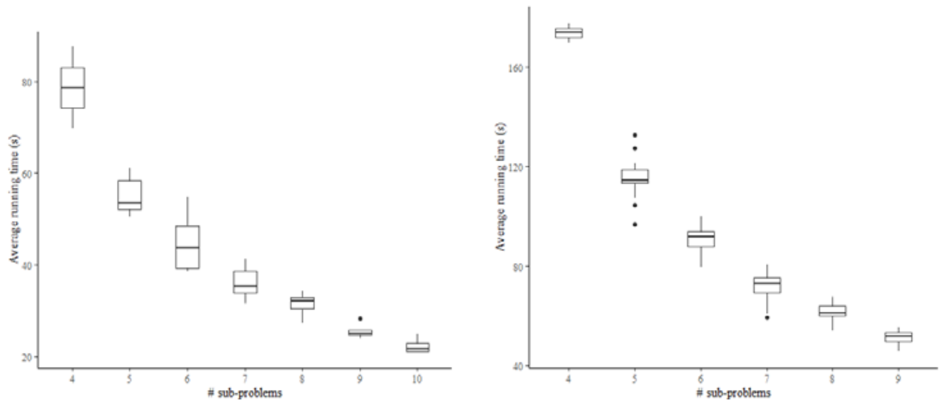
Instance	Region	Day	Day part	#Vehicles	#Customers	#Depots
A	A	Monday	morning	289	1303	7
B	B	Saturday	morning	329	2962	3
C	C	Wednesday	afternoon	345	2676	6
D	D	Monday	afternoon	299	2233	5

Sub-problem size

The size fitness function contains the parameters $\alpha, \beta, \gamma, \delta$ and ζ. We start with setting the coefficients such that all measures are about equally represented. This gives the following values: $\alpha = 5, \beta = 1, \gamma = 0.1, \delta = 2.5$, and $\zeta = 150$. Next, we change the value of δ and ζ to change the sub-problem sizes, keeping α, β, and γ constant, since these parameters do not represent the sub-problem size. Furthermore, we let the minimum number of sub-problems range between 4 and 7 and for the maximum number of sub-problems we use the values 5, 8 and 10. We only consider the combinations for which the minimum is smaller than the maximum number of sub-problems. Results are obtained by running our algorithm for $r_1 = 75$ iterations for all combinations, using only the size fitness function.

With the parameter ζ we try to obtain the desired number of sub-problems. We expect that the percentage between the minimum and maximum allowed sub-problems increases when ζ increases, since deviations are penalized more severely. Only in some cases, the number of sub-problems is not within the limits, suggesting that even with small weights the algorithm is able to create the desired number of sub-problems. Almost all violations occur for combinations with a minimum number of sub-problems of 4, a maximum of 5 and ζ smaller than 175. In each iteration a new chromosome is created by inserting a sub-problem and only with mutation, the number of sub-problems can be lowered. Therefore, the algorithm might have difficulty with creating a low number of sub-problems, especially when violations only have a relative small penalty.

The goal of increasing the number of sub-problems is to decrease the running time. In Figure 3 we show the running times for instance D using both templates. These figures show that the running time decreases if the number of sub-problems increases. Note that these running times only involve solving the sub-problems. The splitting time is not influenced by these parameter values, and is therefore omitted from this analysis. For the slow template, the running time seems to decrease exponentially with the number of sub-problems. This is expected, since there are elements in the slow template that increase the running time exponentially with the number of customers. For the fast template the decrease seems more linear. The course of the running times for the other instances is similar.



(a) (b)

Figure 3: Distribution of the running time (in seconds) for the number of sub-problems using the fast template (a) an the slow template (b).

Since all sub-problems are solved in parallel, the total running time is defined by the slowest sub-problem. We therefore also aim at an equal distribution of customers over the sub-problems, to lower the difference in running time.

Figure 4 shows the distribution of the standard deviation of the number of customers per sub-problem for each value of δ. As expected, the standard deviation decreases if the value of δ increases, since we penalize large standard deviations more severely. However, the minimum value does not decrease. If the standard deviation is really low, the value of delta does not matter much, since its contribution to the total fitness function is low anyway. If $\delta = 1.5$, the other measures have a relatively large weight compared to the standard deviation. Hence, a chromosome with a bad (high) standard deviation can still be the best if the value of the other measures is low. For larger values of δ this is less likely, explaining the decrease in maximum standard deviation. Note that Figure 4 is independent of the solving template that is used.

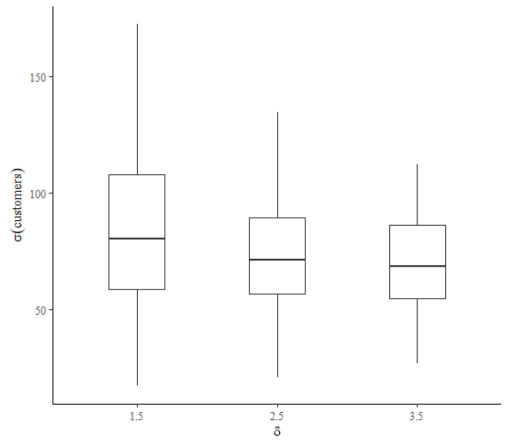
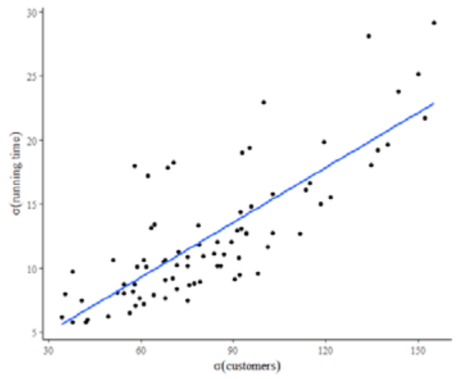


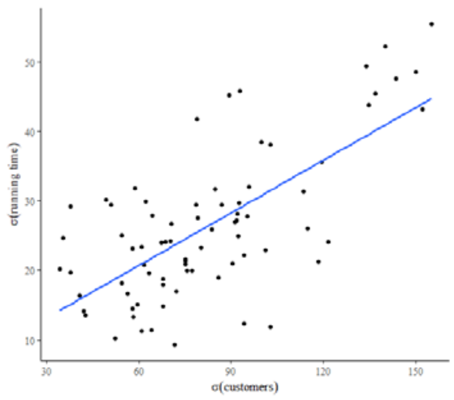
Figure 4: Distribution of the standard deviation of number of customers (σ) for each value of δ.

Next we check if a lower standard deviation in the number of customers also results in a decrease in the standard deviation and the maximum of the running times. In Figure 5 the relation between the standard deviation of the number of orders and the standard deviation of the running times (in seconds) is shown. We clearly see a lower running time if the standard deviation of the number of orders decreases. For the slow template the relation is less strong. The running time of a template is not solely defined by the number of customers, and for more complex templates, like the slow template, the running time might depend more on other factors, such as the number of depots or distances between customers.

At first sight, the maximum running time does not decrease if customers are more evenly distributed. This is because the maximum running time is also partly dependent on the number of sub-problems, because more sub-problems result in a lower average running time. The standard deviation of the number of customers does not necessarily decrease if the number of sub-problems increases. If we only compare the maximum running time for the same number of sub-problems, we now see that the maximum running time decreases if the standard deviation of the number of orders decreases.



(a)



(b)

Figure 5: Relation between the standard deviation of the number of orders and the standard deviation of the running time (in seconds) using the fast (a) and slow (b) template.

The results presented in this section show that we can guide the sub-problems size towards the desired size. The running time can be reduced if the number of sub-problems is increased. Since the total running time is defined by the slowest sub-problem, the running time can be further reduced if the sub-problems are more equal in size. We determine the parameter values based on this information.

The running times for the fast template are all relatively low, hence we considered the objective values to see if we can make a comparison between parameter combinations. There is a high variability in costs, as simple templates tend to find good solutions in some cases and very bad in others. We did not find a link between the cost and the parameter values. We therefore choose a random combination from the best 30%, which can be found in Table 4. With this combination, the number of sub-problems is not heavily restricted and we allow for more variation in the sub-problem sizes. For completeness we also added the values of the other (constant) parameters.

Table 4: Chosen parameter values for each template.

	δ	ζ	min	max	α	β	γ
Fast template	1.5	150	4	10	5	1	0.1
Slow template	2.5	150	7	10	5	1	0.1

The difference in objective values for the slow problem is smaller, as this template searches more thoroughly for a good solution. We therefore consider the running time in choosing the parameters. For this combination we choose a parameter combination which results in more sub-problems, to lower the running time more. The value for δ is higher than for the fast template, to lower the difference in sub-problem size. The chosen combination can also be found in Table 4.

Approximate routing cost

We will now discuss the tuning of the approximate cost fitness function. We estimate the coefficients of this function with a linear regression, with the costs being the dependent variable and the quality measures the independent variables. It is possible that the estimates of the coefficients differ between solving methods. We therefore need different models to approximate the routing costs from the two templates.

The data that is used in the linear regression is created as follows. For each instances in Table 3 we run our algorithm for $r_1 = 100$ iterations using the size fitness function, giving us 100 data points plus 25 points from the initialization procedure per parameter combination. To lower the dependence on the parameter values of the size fitness function we run the algorithm with several parameter combinations. We do not use the approximate cost function yet, since we then need to initialize the coefficients, which might influence the estimates. Because there are more unplanned customers in the beginning of our algorithm, we decided to perform 100 instead of 75 iterations to extend our training set.

For each created chromosome we save the value of the independent variables and obtain the routing cost by solving all sub-problems using CVRS. We removed observations for which there were unplanned customers, since these solutions are not feasible and the routing cost is not representative. From the remaining observations we randomly select 70% of the chromosomes for our training set, and the other 30% is used to test our model.

To measure the quality of the approximation we use the in-sample and out-of-sample Mean Percentage Error (MPE) and the Mean Absolute Percentage Error (MAPE). They are defined as follows:

$$MPE = \frac{1}{k}\sum_{s=1}^{k}\frac{C_s - \hat{C}_s}{C_s}, \tag{8}$$

$$MAPE = \frac{1}{k}\sum_{s=1}^{k}\frac{\left|C_s - \hat{C}_s\right|}{C_s}, \tag{9}$$

where C_s denotes the costs for chromosome s obtained by the solution method, $\hat{C}_s$ the approximation of the regression model, and k the number of observations.

Table 5 shows the model estimates and errors for the two templates. Both forward and backward stepwise regression were performed. The threshold for including or removing a variable is set to 0.1. Additional to the variables discussed in Section 5.2.2, we also included the number of customers, n, as a potential variable to account for the difference in sub-problem size.

Table 5: Model estimates using forward stepwise regression (Model 1) and backward stepwise regression (Model 2).

	Fast template		Slow template	
Variables	Model 1	Model 2	Model 1	Model 2
Intercept	333995.80 ***	338326.15 ***	54278.96 ***	55858.26 ***
n	301.85 ***	302.03 ***	79.40 ***	62.72 ***
SumP				
VarP		-22075.20 .		
MinP				
MaxP	-1751.13 *	-2812.31 *		
SumMinP				
SumMaxP				
SumM				
VarM		148721.16 *		
MinM				
MaxM				
SumD				
VarD	28774.64 .			
SumTW	18.25 ***	21.12 ***	19.28 ***	24.20 ***
VarTW	65.37 ***	67.88 ***		
SumOverlap	0.11		-0.05 ***	-0.05 ***
AvgOverlap				
VarOverlap			18.40	
TotD/AvCap	-4261.82 ***	-4324.54 ***	1651.89 ***	1809.56 ***
AvCap/AvD	-368.19 ***	-424.50 ***		
VarX $\times$ VarY	1356228.00 ***	1522269.75 ***		
R2 (adjusted)	0.9698	0.9698	0.9289	0.9289
MPE (in-sample)	-0.15%	-0.15%	-0.23%	-0.23%
MAPE (in-sample)	2.88%	2.87%	3.07%	3.07%
MPE (out-of-sample)	-0.49%	-0.38%	-0.40%	-0.43%
MAPE (out-of-sample)	3.12%	3.04%	2.92%	2.94%

***: $p < 0.001$; **: $p < 0.01$; *: $p < 0.05$; .: $p < 0.1$.

As expected, the model estimates for the fast template and the slow template differ. Different variables are included, and the size and sign of common variables are different. The interpretation of the sign of some variables is not intuitively clear, for example for MaxP. One would expect a higher cost if customers are further apart, but the reverse is true. A possible explanation for this is that instance A spans a larger area resulting in larger values for MaxP. At the same time, this instance has lower costs due to the lower number of customers, possibly causing the mentioned effect.

As for the performance, the R2 of the slow templates is lower, but the performance in terms of in-sample and out-of-sample is comparable. The performance of the forward and backward stepwise regression models do not differ much. Since the difference is small, we decided to only consider backward stepwise regression from now on.

Our MPE is small and thus suggests that the errors are almost evenly distributed around zero. Although the absolute errors (MAPE) are also relatively small, our algorithm requires that the ranking of the fitness values of the chromosomes is comparable with the ranking of the costs. We therefore also checked the average deviation of the rank. Since the costs considerably differ per region and parameter combination, we determined the per region and per parameter combination to make a fair comparison.

If we analyse these deviations, we see that the model is, in most cases, not able to predict a comparable ranking. If we look at the distribution of customers in Figure 2, we see that the distributions of customers are different, both in shape and size. As a result, the relation between the costs and the variables might differ. We therefore also estimate the coefficients for each region separately, to see if there is a significant difference between the regions. The model estimates are represented in Table 6.

The table shows that a different set of variables is significant for each region, indicating that the we need different models. Although the R2 is lower than for the models in Table 5, the errors are considerably smaller. Only the errors for instance A are larger than for the combined model. Maybe this is caused by the low number of customers compared to the other instances. If we estimate the coefficients with another instance of the same region the errors are smaller, however the ranking is still poor. There might thus be other factors not included in the model that influence the costs for this region making prediction more difficult. We can rule out the solution method, as the other template does not have a better performance.

The distribution of the customers can make it more difficult to deduce the cost from the problem decomposition. Customers are gathered in small clusters and there are even some customers that are separated from the other customers. Not having the right clusters together can result in high costs, but it is difficult to determine the right combination. Due to the already high distances between clusters, the distance related measures do not differ much between chromosomes, while there is high variability in the costs. For the time window related measures the variability is high, but we do not see a relation with the cost. Because neighbouring customers have subsequent time windows, it is possible to visit them in a short time span and keeping the distance low. Short time windows and little overlap thus does not necessarily increase the cost.

Since the models for each region separately perform better, we decide to continue our algorithm with these models.

Table 6: Model estimates per region using backward stepwise regression.

Variables	Fast template	Slow template
Intercept	493538.30 ***	374720.52 ***
SumP		
VarP	-99212.19 ***	26221.32 *
MinP	418862.40 ***	
MaxP		
SumMinP	-6765.97 *	
SumMaxP	-34.81 **	
SumM		
VarM	492932.20 ***	-387.76 **
MinM	418862.4 ***	-140098.58 .
MaxM		
SumD		165.13 .
VarD	-45770.61 ***	28345.73 **
VarTW		101.28 **
SumOverlap	-0.12 **	
AvgOverlap		
VarOverlap		-198.83 *
TotD/AvCap		
AvCap/AvD		
VarX × VarY	1856352 ***	1114504.20 **
R2 (adjusted)	0.2998	0.0819

Variables	Fast template	Slow template
Intercept	737484.46 ***	794484 ***
SumP		
VarP		
MinP		
MaxP	-48576.21 ***	-52833.88 ***
SumMinP		46684.15 **
SumMaxP		87.41 .
SumM		
VarM		
MinM	6817801.23 **	
MaxM	69667.45 **	
SumD		-1060.18 **
VarD		1302691 .
VarTW	1066.23 **	
SumOverlap		-0.08 **
AvgOverlap	-135068.23 *	
VarOverlap	-1184.57 .	
TotD/AvCap		
AvCap/AvD	8307.83 .	
VarX × VarY		
R2 (adjusted)	0.1833	0.2014

Variables	Fast template	Slow template
Intercept	670353.5 ***	642676.60 ***
SumP	-1.19 *	-0.46 ***
VarP	-145280.7 *	101662.50 *
MinP		
MaxP		
SumMinP	-11961.88 *	-10888.95 *
SumMaxP		
SumM	731.48 *	
VarM	328515.2 .	-353110.10 .
MinM	3484730 **	
MaxM		
SumD		
VarD	466020.2 ***	415069.20 ***
VarTW	569.80 **	-156.03 .
SumOverlap	0.08 .	
AvgOverlap	-21743.46 *	
VarOverlap	-582.10 **	243.72 .
TotD/AvCap		
AvCap/AvD	1120.07 .	
VarX × VarY		4398313 **
R2 (adjusted)	0.2611	0.3068

Variables	Fast template	Slow template
Intercept	19699220 ***	5649141 ***
SumP	-3.64 *	0.71 .
VarP		
MinP		
MaxP		
SumMinP		
SumMaxP		-46.70 **
SumM	2153.66 *	
VarM		
MinM		
MaxM		
SumD		
VarD	-22015710 **	
VarTW		
SumOverlap	0.21 **	-0.09 ***
AvgOverlap		
VarOverlap		
TotD/AvCap	-175204.4 ***	-46293.90 ***
AvCap/AvD	943.59 ***	
VarX × VarY	-25149490 *	
R2 (adjusted)	0.6495	0.3300

6.2.3 Results

In this section, we present the solutions of our algorithm on the described instances. A complete description of the instances can be found in Table 10 in Appendix B. For comparison we also show the solution quality and running time of the command templates when no decomposition is used.

In Table 7 the costs and running times for both the fast and the slow template are shown for indication. The costs of the slow template are on average 20% lower, but the fast template is on average more than three times faster. We clearly see a trade-of here between solution quality and running time.

Table 7: Plan cost and running time of the fast template for all instances without decomposition.

	Fast template		Slow template	
Instance	Cost	Time	Cost	Time
A1	415,414.0	00:06:54	314,750.6	00:18:35
A2	685,485.2	00:13:55	508,290.1	00:59:13
A3	616,204.6	00:12:06	424,613.2	00:44:52
A4	697,847.4	00:14:45	489,931.4	00:47:57
A5	683,200.3	00:14:38	502,246.8	00:53:25
B1	484,786.3	00:07:42	450,571.8	00:22:42
B2	751,335.9	00:22:08	587,052.0	01:29:19
B3	653,507.2	00:18:15	569,445.9	00:58:31
B4	677,793.7	00:18:20	581,130.9	01:11:14
B5	703,573.7	00:19:13	615,122.6	01:06:29
C1	253,146.5	00:04:23	224,042.5	00:14:22
C2	658,910.8	00:17:12	508,507.6	01:04:50
C3	616,701.2	00:17:36	440,736.8	01:02:20
C4	756,849.6	00:20:29	540,695.2	01:12:50
C5	720,107.4	00:19:00	572,803.6	01:13:26
D1	586,932.9	00:09:18	479,525.5	00:27:20
D2	609,521.0	00:13:29	442,798.8	00:54:12
D3	619,545.1	00:13:51	449,872.1	00:49:34
D4	569,170.2	00:11:15	460,789.4	00:41:22
D5	553,738.1	00:10:10	457,648.6	00:38:05
Avg.	615,537.5	00:14:24	481,028.8	00:51:32

We now run our algorithm for three multi-process iterations to see the effect on the running time and solution quality. In Table 8 the results are shown as the percentage deviation from the results in Table 7. To get a clearer picture, we averaged the results per region. The results per instance can be found in Table 14 and 15 in Appendix C.

Table 8: Average deviation of the cost and running time per region for the fast and slow template. The denoted values are compared to solutions obtained without splitting.

(a) Results for the fast template.

	1 iteration		2 iterations		3 iteration	
Region	Cost	Time	Cost	Time	Cost	Time
A	+4.5%	-72.0%	+0.9%	-61.1%	-1.5%	-52.1%
B	+1.1%	-68.6%	-4.5%	-52.5%	-6.7%	-44.3%
C	+7.3%	-69.4%	+0.8%	-61.4%	-0.9%	-51.0%
D	-3.3%	-72.8%	-6.8%	-62.1%	-8.3%	-52.5%

(b) Results for the slow template.

	1 iteration		2 iterations		3 iteration	
Region	Cost	Time	Cost	Time	Cost	Time
A	+11.0%	-86.6%	+5.5%	-83.0%	+1.9%	-79.6%
B	+6.7%	-87.8%	+6.5%	-84.3%	-1.9%	-80.8%
C	+5.6%	-86.8%	-0.4%	-83.8%	-2.1%	-80.7%
D	+2.2%	-89.1%	-2.2%	-85.7%	-4.6%	-83.6%

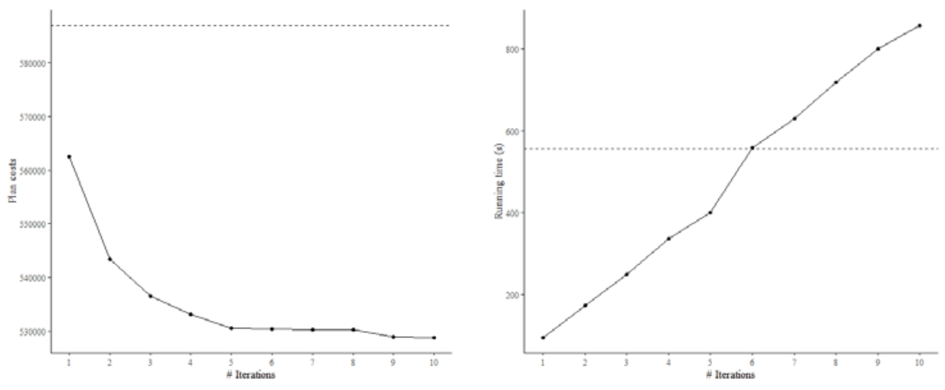
From these results we can see that we achieved our first goal of reducing the running time. For the fast template we can obtain solutions in half the time, while for the slow template we can reduce the running time even more. If we increase the number of iterations for the fast template, the running time will soon approach the running time without splitting. For the slow template, there is more room for performing more iterations, since the running time is still reduced by 80% after three iterations. From the second iteration the splitting time is much lower than for the first iteration, because we now split on routes instead of single customers, giving us less elements to consider. The solving time also decreases, since the construction phase can be skipped.

The cost are still considerably higher after the first iteration, with the exception of the fast template for region D. For most instances, however, the costs decrease each iteration, pointing out the usefulness of multiple iterations. The slow template needs more iterations to obtain a sufficient cost, but after three iterations, the average cost for almost all regions is lower than without splitting. We can thus not only obtain solutions in a shorter amount of time, but we can also improve the costs.

Note that these results show the average per region, therefore, not all instances perform as well. If we look at more detail at the individual instances of region A, we see that instance A4 and A5 have a positive deviation after three iterations, meaning higher costs. Because these instance have longer time windows and more overlap, a different model might be needed.

To see if the current approximate cost function causes the high cost, we ran the algorithm again using models fitted on the instance itself. We indeed obtain better results, suggesting that different variables are significant for these specific day parts. For the slow template the same results apply for instance A4. For instance A5, however, the regular model estimates give sufficient results. More data is therefore needed to determine if these day parts differ from the others. For the other regions, the results do not differ much per instance.

We also performed more multi-process iterations to see if we can even further reduce the costs. In Figure 6 we show the cost and running time (in seconds) for 10 iterations for instance D1. For comparison we also added the cost and running time of the template without decomposition. In the beginning the cost still decreases each step, but after 5 iterations we we see that the cost flattens, because there is less diversification in the splits.

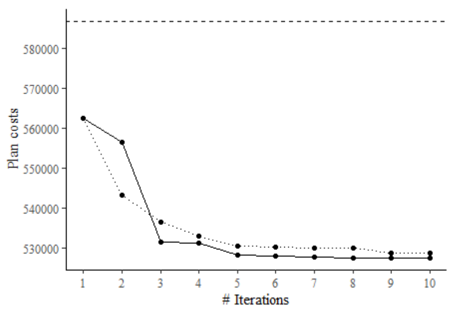


(a) (b)

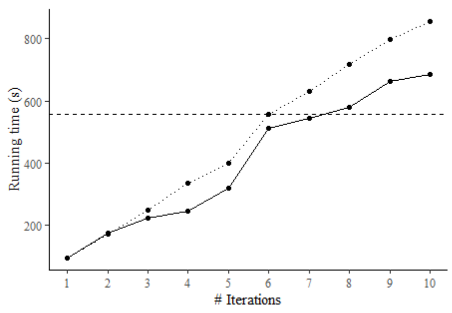
Figure 6: Course of the plan cost (a) and total running time (b) for instance D1. The dashed lines denote the cost and running time without decomposition.

To create more diversity in our splits, we combine our genetic algorithm with two methods of ORTEC. These methods divide the customers and vehicles equally over the sub-problems. The methods implemented at ORTEC are the Sweep method and the distance ordering method. The Sweep method is already described in the literature review. In the distance ordering method, the customer farthest from the depot is selected to form a sub-problem. The closest customers are added until the sub-problem has its proportion of the customers. This is repeated with the customers that are left until all customers are assigned to a sub-problems. The vehicle are then equally divided over the sub-problems.

For both methods, we need to determine the number of sub-problems beforehand. Each time a method is used, we draw a random number between 6 and 10 for the number of sub-problems. The Sweep method is used in iteration 3, 6 and 9, and the distance methods is used in iteration 2, 5, and 8. In the remaining iterations we use our genetic algorithm.



(a)



(b)

Figure 7: Course of the plan cost (a) and total running time (b) for a combination of splitting methods for instance D1. The dashed lines denote the cost and running time of the template without decomposition. The dotted lines denote the values when only our genetic algorithm is used.

In Figure 7 the course of the costs and running time for the described combination of splitting methods is used. The dotted lines denote the values from Figure 6. We can see that this combination has a lower cost and total running time. The performance can thus be improved by generating more diverse splits. The costs, however, still converge after a few iterations. This might suggest that the solving template is not able to find a better solution.

We also ran the mentioned combination of splitting methods with the slow template. We obtain similar results. The costs for this template also converge, mostly after 6 iterations. The running time is, however, still way below the running time without decomposition.

6.3 Theoretical case

We also want to test our method on other instances, to see if we obtain similar conclusions as with our practical case.

6.3.1 Problem instances

The problem instances for the theoretical case are based on the Solomon benchmark instances. These instance are suitable to test methods for the capacitated vehicle routing problem with time windows. The objective is to minimize the number of vehicles and total distance. This set of instances consists of six different types, based on the distribution of customers and the length of time window. There are three types of distributions, namely uniformly distributed customers (R), cluster customers (C) and a combination of both types (RC). The time window type are split in short (1) and long (2) time windows. Furthermore, the instances consist of 25, 50 or 100 customers.

We select the following three instances, all containing 100 customers: RC106_100, RC107_100, RC108_100, R106_100, R107_100, R108_100, and C106_100, C107_100, C108_100. Since our method is designed for larger problem, we modified the instances as follows. Each customer is used to simulate a new customer by adding a random value from a $N(0, 2)$ distribution to its coordinates. The capacity, service duration and time window are randomly drawn from the available values. We simulate 15, 20 or 25 points per customer, giving us in total 1500, 2000 or 2500 customers per instance.

6.3.2 Results

For the theoretical case we use the same base configuration as for the practical case (Section 6.1). For the fitness function we will use the following parameter values. For the size fitness function we use the same parameter values for the fast template, see Table 4. The coefficients for the approximate cost function are taken from Nicola et al. (2019), where the coefficients are estimated for nearest neighbour solutions.

The construction phase of our solution templates resembles the nearest neighbour solution method. We will therefore only use the construction phase in the solving template for the theoretical case in the first iteration. Since the construction phase only inserts unplanned orders, we also do local search from the second iteration, as the costs would not be changed otherwise.

Table 9 shows the results for the theoretical case. For the practical case we concluded that the gain in running time was smaller for the fast template. The first splitting iteration is relatively slow compared to the solving time. The solving time of is now even lower, resulting in less reduction of the running time when using multi-process optimization.

Table 9: Plan cost and running time after three iterations for the theoretical instances.

	No decomposition		1 iteration		2 iterations		3 iterations	
	Cost	Time	Cost	Time	Cost	Time	Cost	Time
C106_1500	197,659.9	04:23.23	+1.8%	-53.6%	+0.5%	-22.1%	+0.3%	+10%
C107_2000	248,853.7	10:03.04	+2.5%	-42.8%	+1.2%	-24.3%	+1.1%	-5.4%
C108_2500	299,660.4	11:09.55	+1.7%	-34.9%	+0.5%	+3.6%	+0.3%	+32.1%
R106_1500	213,813.8	05:20.59	+4.5%	-51.9%	+3.5%	-24.8%	+3.3%	+1.4%
R107_2000	246,526.9	10:13.53	+2.7%	-55.7%	+1.3%	-30.0%	+1.1%	-6.1%
R108_2500	291,890.1	13:50.90	+0.7%	-47.8%	+0.0%	-18.4%	-0.7%	+7.6%
RC106_1500	203,527.4	04:41.21	+5.7%	-56.4%	+4.4%	-32.2%	+4.3%	-5.6%
RC107_2000	248,570.9	07:13.34	+4.3%	-50.1%	+3.6%	-23.0%	+3.5%	+2.0%
RC108_2500	300,872.5	14:06.27	+3.8%	-53.2%	+2.5%	-29.2%	+2.1%	-6.7%

For most cases the costs are also higher. This can be due to the coefficients of the approximate cost fitness function. For the RC model, the estimation errors were considerably larger than for the other models, which is reflected in the large (positive) deviation in the cost for this group. We also do not know if the estimates would change if they were estimated on the simulated customers.

7 Conclusion

In this thesis we considered a Vehicle Routing Problem with many different attributes. Since the VRP is a NP-hard problem, large sized problems can not be solved to optimality within reasonable time. Even the running time of approximate solution techniques can increase rapidly if the problem size increases or if the problem becomes more complex. For many real world problems, the construction of routes need to be fast due to increasing customers expectations. This results in a trade-of between running time and solution quality.

Recently, ORTEC started investigating a new method to overcome this problem, called multi-process route optimization. This method first splits the large VRP over smaller sub-problems and then solves these sub-problems simultaneously using multiple processors. The goal is to obtain a solution of similar quality in substantially less time. Our method extends this approach by also incorporating the attributes to create decompositions. With a genetic algorithm we determined the best decomposition. Furthermore, we investigated how we need to adjust the fitness function if different solving methods are used.

We tuned our algorithm for two different solving templates, differing in both the running time and solution quality. The first template produces high quality solutions at the cost of a longer running time, while the second template solves the problem much faster but also has a lower quality solution. Our algorithm consists of two phases with two different fitness functions, where we first focus on the number of sub-problems and their sizes and then focus on obtaining a good objective value.

We saw that the average running time for both templates decreases substantially if the number of sub-problems increase. Especially the slow template can benefit from a reduction in sub-problem size. We therefore chose to create more sub-problems for this template. We noticed that different measures were important in estimating the routing cost for both templates, resulting in different fitness functions.

The computational results on the practical case showed that solutions can be obtained in substantially less time if the problem is decomposed in sub-problems, without loss of quality. Even more, in most cases we were able to obtain even better solutions. For the fast template we need to keep the number of iterations relatively low, to preserve the reduction in running time. The slow template needs more iterations to obtain a similar (or lower) cost, but the reduction in running time is still significant.

The solution method for the theoretical case is much faster. Since the running time of the first iteration of our algorithm is relatively slow, the reduction in running time is less than for the practical case. This suggest that our algorithm is more beneficial when slow and complex solution methods are used.

Our approach thus makes it possible to solve instances that could not be solved before, because they are too large. It also makes it possible to use more complex templates, that are normally too slow on large cases. Because complex templates often have more components that increase the running time exponentially with the number of customers.

Although we need to tune the parameters for new types of instances, our method can be promising for the considered retail company. Instances on the same day part and in the same region are very similar. Hence, we only need to tune the parameters once. Since there are also some differences between days, one can choose to tune the parameter per day (part) and per region.

Although the method described in this thesis performs well, we propose other interesting ideas and extensions that can be investigated in future research. First of all we saw that after a few iterations the costs converged and did not decrease any more. To create more diversity in the splits and hopefully avoid convergence, we combined our algorithm with other methods implemented by ORTEC. We saw, however, that the costs still converged after a few iterations. Other splitting methods might be used or different combinations to check whether they result in splits where more improvement can be found.

Secondly, we saw that our approximate cost fitness function was in some cases not able to correctly predict the real cost. More complex models or different factors can be included to improve the fitness function. We did not include any variables relating to the heterogeneous fleet. The approximate cost fitness function will probably benefit from these kind of variables.

We also saw that the parameter values of the size fitness function sometimes direct the solution in the wrong direction. Better tuning of these values might therefore be necessary. Furthermore it might be interesting to make our method more dynamic, for example by choosing a command template based on the sub-problem size or updating the parameter values during the algorithm.

References

Anderberg, M. R. (1973). *Cluster Analysis for Applications*. Academic press.

Barreto, S., Ferreira, C., Paixao, J., and Santos, B. S. (2007). Using clustering analysis in a capacitated location-routing problem. *European Journal of Operational Research*, 179(3):968–977.

Beasley, J. E. (1983). Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408.

Bowerman, R. L., Calamai, P. H., and Hall, G. B. (1994). The spacefilling curve with optimal partitioning heuristic for the vehicle routing problem. *European Journal of Operational Research*, 76(1):128–142.

Cheng, C.-B. and Wang, K.-P. (2009). Solving a vehicle routing problem with time windows by a decomposition technique and a genetic algorithm. *Expert Systems with Applications*, 36(4):7758–7763.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

Daganzo, C. F. (1984). The distance traveled to visit n points with a maximum of c stops per vehicle: An analytic model and an application. *Transportation science*, 18(4):331–350.

Dondo, R. and Cerdá, J. (2007). A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research*, 176(3):1478–1507.

Figliozzi, M. A. (2008). Planning approximations to the average length of vehicle routing problems with varying customer demands and routing constraints. *Transportation Research Record*, 2089(1):1–8.

Flood, M. M. (1956). The traveling-salesman problem. *Operations research*, 4(1):61–75.

Ganesh, K. and Narendran, T. (2007). Cloves: A cluster-and-search heuristic to solve the vehicle routing problem with delivery and pick-up. *European Journal of Operational Research*, 178(3):699–717.

Geetha, S., Poonthalir, G., and Vanathi, P. (2009). Improved k-means algorithm for capacitated clustering problem. *INFOCOMP*, 8(4):52–59.

Gillett, B. E. and Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 22(2):340–349.

Ho, W., Ho, G. T., Ji, P., and Lau, H. C. (2008). A hybrid genetic algorithm for the multi-depot vehicle routing problem. *Engineering Applications of Artificial Intelligence*, 21(4):548–557.

Jorgensen, R. M., Larsen, J., and Bergvinsdottir, K. B. (2007). Solving the dial-a-ride problem using genetic algorithms. *Journal of the operational research society*, 58(10):1321–1331.

Kim, B.-I., Kim, S., and Sahoo, S. (2006). Waste collection vehicle routing problem with time windows. *Computers & Operations Research*, 33(12):3624–3642.

Koskosidis, Y. A., Powell, W. B., and Solomon, M. M. (1992). An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation science*, 26(2):69–85.

Labadi, N., Prins, C., and Reghioui, M. (2008). A memetic algorithm for the vehicle routing problem with time windows. *RAIRO-Operations research*, 42(3):415–431.

Lacomme, P., Prins, C., and Ramdane-Chérif, W. (2001). A genetic algorithm for the capacitated arc routing problem and its extensions. In *Workshops on Applications of Evolutionary Computation*, pages 473–483. Springer.

Lenstra, J. K. and Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

Maulik, U. and Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique. *Pattern recognition*, 33(9):1455–1465.

Nicola, D., Vetschera, R., and Dragomir, A. (2019). Total distance approximations for routing solutions. *Computers & Operations Research*, 102:67–74.

Ostertag, A. (2008). *Decomposition strategies for large scale multi depot vehicle routing problems*. PhD thesis, University of Vienna.

Pankratz, G. (2005). A grouping genetic algorithm for the pickup and delivery problem with time windows. *Or Spectrum*, 27(1):21–41.

Potter, T. and Bossomaier, T. (1995). Solving vehicle routing problems with genetic algorithms. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 788–793. IEEE.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002.

Prins, C., Lacomme, P., and Prodhon, C. (2014). Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200.

Reed, M., Yiannakou, A., and Evering, R. (2014). An ant colony algorithm for the multi-compartment vehicle routing problem. *Applied Soft Computing*, 15:169–176.

Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591.

Ryan, D. M., Hjorring, C., and Glover, F. (1993). Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, 44(3):289–296.

Salhi, S. and Sari, M. (1997). A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. *European Journal of Operational Research*, 103(1):95–112.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B: Methodological*, 44(2):246–266.

Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673.

Taillard, É. D. and Voss, S. (2002). Popmusic—partial optimization metaheuristic under special intensification conditions. In *Essays and surveys in metaheuristics*, pages 613–629. Springer.

Tansini, L. and Viera, O. (2006). New measures of proximity for the assignment algorithms in the mdvrptw. *Journal of the Operational Research Society*, 57(3):241–249.

Thangiah, S. R. (1993). *Vehicle routing with time windows using genetic algorithms*. Citeseer.

Thangiah, S. R. and Salhi, S. (2001). Genetic clustering: an adaptive heuristic for the multidepot vehicle routing problem. *Applied Artificial Intelligence*, 15(4):361–383.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2014). Implicit depot assignments and rotations in vehicle routing heuristics. *European Journal of Operational Research*, 237(1):15–28.

A List of Symbols

Below we give an overview of all used symbols and notations.

Network graph

G	directed graph representing the network graph
V	set of all vertices in the network
A	set of all arcs in the network
N	set of customer nodes in the network
D	set of depot nodes in the network

Customer

N	set of all customers
N_r	set of all customers in sub-problem r
$[a_i, b_i]$	time window of customer $i \in N$
s_i	service duration of customer $i \in N$
$P_{1,i}$	demand quantity of customer $i \in N$
$P_{2,i}$	demand quantity of customer $i \in N$

Depot

D	set of all depots
D_r	set of all depots in sub-problem r

Vehicle

K	set of all vehicles
K_d	set of vehicles available at depot d
K_r	set of vehicles in sub-problem r
W	set of vehicle types
Q_w	capacity of vehicle type $w \in W$
s_w	driving speed of vehicle type $w \in W$
o_w	set of operating cost of vehicle type $w \in W$

Genetic algorithm

S	chromosome representing set of sub-problems
$F(S)$	fitness function of chromosome S representing the size
C_S	routing cost of chromosome S
z_r	centre of gravity of sub-problem r
r_1	number of iterations with size fitness function
r_2	number of iterations with approximate cost fitness function

k	initial number of sub-problems
p_m	mutation probability of mutation operator m
x	number of customers reassigned by the mutation operator
y	number of vehicle reassigned by the mutation operator
n	number of individuals in the population
z	number of worst individuals in population

B Problem instances in detail

In Table 10 a more detailed description of the used instances can be found.

Table 10: Description of the instances from the retail company

instance	region	weekday	day-part	#vehicles	#orders	#depots
A1	A	Monday	morning	289	1303	7
A2	A	Monday	afternoon	291	2286	7
A3	A	Wednesday	morning	293	2089	7
A4	A	Wednesday	afternoon	293	2287	7
A5	A	Saturday	morning	293	2275	7
B1	B	Monday	morning	322	1861	2
B2	B	Monday	afternoon	329	3099	3
B3	B	Wednesday	morning	329	2876	3
B4	B	Wednesday	afternoon	329	2805	3
B5	B	Saturday	morning	329	2962	3
C1	C	Monday	morning	333	1046	6
C2	C	Monday	afternoon	339	2481	6
C3	C	Wednesday	morning	345	2446	6
C4	C	Wednesday	afternoon	345	2676	6
C5	C	Saturday	morning	345	2570	6
D1	D	Monday	morning	308	1840	5
D2	D	Monday	afternoon	299	2233	5
D3	D	Wednesday	morning	303	2238	4
D4	D	Wednesday	afternoon	300	2093	4
D5	D	Saturday	morning	299	1862	4

In Table 11 we give an overview of the different vehicles types per instance.

Table 11: Number of vehicle types per instance.

Instance	#Vehicles type1	#Vehicles type2	#Vehciles type3	#Vehicles type4	#Vehicles type5
A1	0	11	111	150	17
A2	0	11	112	150	18
A3	0	11	113	150	19
A4	0	11	113	150	19
A5	0	11	113	150	19
B1	0	12	160	150	0
B2	0	13	166	150	0
B3	0	13	166	150	0
B4	0	13	166	150	0
B5	0	13	166	150	0
C1	0	14	157	150	12
C2	0	14	163	150	12
C3	0	14	169	150	12
C4	0	14	169	150	12
C5	0	13	170	150	12
D1	2	22	132	150	0
D2	4	22	125	150	0
D3	3	22	126	150	0
D4	5	22	125	150	0
D5	2	22	125	150	0

C Results

Table 12: Number of sub-problems and standard deviation of the number of orders per sub-problem for some parameter combinations.

A		B		C		D_1					
#	σ	#	σ	#	σ	#	σ	δ	ζ	min	max
7	91.51	3	17.04	5	63.41	4	37.75	1.5	125	4	5
10	75.31	6	75.50	10	92.43	8	114.91	1.5	125	4	10
9	50.35	8	50.87	10	65.46	8	60.92	1.5	125	7	10
8	70.84	6	71.95	8	62.98	7	118.38	2.5	150	4	8
7	73.33	7	87.86	7	53.53	7	79.07	2.5	175	6	8
10	64.06	7	52.82	10	81.72	10	97.91	3.5	150	6	10
7	69.89	8	91.74	7	64.91	7	42.06	3.5	175	6	8
7	63.62	8	60.68	8	87.35	8	86.92	3.5	175	7	8

Table 13: Maximum, average and standard deviation of the sub-problem running times in minutes for instance D for some parameter combinations.

Fast template			Slow template						
Max	Avg	Sd	Max	Avg	Sd	δ	ζ	min	max
01:28.71	01:18.66	00:08.74	03:22.68	02:49.95	00:29.15	1.5	125	4	5
00:48.56	00:31.55	00:13.48	01:28.21	00:59.56	00:25.99	1.5	125	4	10
00:46.92	00:35.39	00:08.02	01:20.72	01:03.99	00:11.23	1.5	125	7	10
00:52.98	00:34.91	00:15.03	01:58.60	01:19.19	00:21.19	2.5	150	4	8
00:46.84	00:34.92	00:11.04	01:38.41	01:02.75	00:27.56	2.5	175	6	8
00:33.08	00:22.35	00:08.01	01:36.72	00:52.18	00:21.45	3.5	150	6	10
00:46.42	00:37.14	00:05.31	01:29.62	01:04.58	00:14.15	3.5	175	6	8
00:41.47	00:30.22	00:10.68	01:35.87	01:05.06	00:29.38	3.5	175	7	8

Table 14: Deviation of the plan costs and running times from the solution obtained without decomposition for three multi-process iterations using the fast template.

	1 iteration		2 iterations		3 iterations	
	Cost	Time	Cost	Time	Cost	Time
A1	+5.7%	-83.9%	-2.5%	-73.6%	-8.3%	-65.4%
A2	+2.5%	-76.9%	+0.4%	-67.1%	-0.2%	-56.8%
A3	+2.3%	-66.1%	-0.6%	-52.8%	-3.4%	-42.6%
A4	+4.6%	-61.1%	+2.9%	-39.5%	+2.8%	-39.5%
A5	+7.6%	-72.1%	+4.6%	-60.9%	+1.8%	-56.4%
B1	+9.4%	-71.5%	+7.4%	-60.1%	+2.2%	-46.3%
B2	-2.6%	-72.2%	-8.3%	-61.9%	-9.3%	-53.8%
B3	+0.1%	-53.5%	-5.6%	-44.3%	-7.0%	-33.8%
B4	-5.4%	-77.7%	-8.0%	-68.7%	-9.8%	-62.4%
B5	+3.3%	-67.8%	-7.6%	-55.4%	-9.5%	-43.3%
C1	+8.0%	-76.7%	+1.8%	-66.9%	+1.6%	-57.0%
C2	+7.5%	-57.8%	+0.6%	-57.2%	-3.9%	-42.3%
C3	+7.9%	-67.3%	-0.2%	-57.2%	-1.0%	-47.5%
C4	+6.9%	-75.6%	+0.9%	-68.1%	-1.7%	-58.8%
C5	+6.3%	-69.5%	+1.0%	-57.6%	+0.3%	-49.6%
D1	-4.2%	-71.4%	-8.4%	-58.5%	-8.3%	-51.4%
D2	-0.3%	-72.0%	-2.4%	-62.4%	-6.5%	-50.7%
D3	+1.3%	-71.5%	-2.5%	-62.8%	-5.0%	-53.8%
D4	-3.6%	-73.8%	-7.5%	-62.6%	-7.7%	-51.8%
D5	-9.6%	-75.6%	-13.4%	-64.5%	-13.7%	-54.6%

Table 15: Deviation of the plan costs and running times from the solution obtained without decomposition for three multi-process iterations using the slow template.

	1 iteration		2 iterations		3 iterations	
	Cost	Time	Cost	Time	Cost	Time
A1	+23.2%	-87.2%	+13.5%	-84.0%	+12.2%	-80.6%
A2	+8.3%	-89.4%	-1.6%	-86.1%	-4.6%	-83.4%
A3	+14.2%	-85.1%	-7.1%	-82.1%	-0.1%	-77.2%
A4	+12.4%	-83.9%	+8.3%	-78.4%	+7.3%	-75.5%
A5	+4.6%	-87.2%	+0.2%	-84.5%	-5.5%	-81.6%
B1	+11.8%	-86.2%	+3.0%	-82.1%	+2.9%	-77.4%
B2	+8.4%	-88.0%	+3.6%	-85.1%	+1.3%	-82.7%
B3	+4.7%	-85.5%	-4.6%	-81.5%	-4.9%	-78.5%
B4	+1.7%	-89.6%	-2.0%	-86.6%	-6.2%	-82.6%
B5	+6.8%	-89.7%	+3.1%	-86.0%	-2.5%	-82.8%
D1	+12.5%	-87.8%	+4.5%	-84.1%	+3.4%	-80.1%
C2	+1.9%	-86.6%	-2.9%	-84.1%	-8.0%	-80.0%
C3	+8.6%	-85.6%	+3.0%	-83.4%	+3.8%	-81.3%
C4	+1.2%	-86.1%	-3.6%	-82.7%	-5.5%	-79.8%
C5	+3.9%	-88.0%	-2.9%	-84.9%	-4.2%	-82.0%
D1	+9.9%	-88.0%	+9.6%	-84.1%	+9.2%	-80.5%
D2	+4.4%	-89.6%	+0.1%	-85.7%	-1.5%	-83.7%
D3	+3.4%	-89.0%	+0.1%	-83.6%	-1.8%	-79.8%
D4	+1.4%	-88.4%	-6.8%	-84.9%	-8.7%	-81.4%
D5	-8.1%	-92.7%	-13.9%	-89.5%	-18.7%	-83.4%