Adaptive Large Neighborhood Search for Rich and Real-World Vehicle Routing Problems

L. M. Simons

Master Applied Mathematics

TU Delft Delft University of Technology

Challenge the future

Adaptive Large Neighborhood Search for Rich and Real-World Vehicle Routing Problems

by

Laura Michelle Simons

April, 2017

A thesis submitted to the
Delft University of Technology
in partial fulfillment of the requirements

for the degree of

Master of Science
in
Applied Mathematics

Supervisor:	Prof. dr. ir. K. Aardal	
Thesis committee:	Prof. dr. ir. K. Aardal,	TU Delft
	Dr. ir. M. van Gijzen,	TU Delft
	Drs. A. Rietveld,	ORTEC B.V.

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

TU Delft Delft University of Technology

ORTEC OPTIMIZE YOUR WORLD

Abstract

In optimization, many variants of the Vehicle Routing Problem exists with all sorts of restrictions and characteristics. Heuristics that find near-optimal solutions are known and tested on in the literature, such as heuristics that escape local minima with the use of large neighborhoods. An example is Adaptive Large Neighborhood Search, which can be used to find near-optimal solutions for rich Vehicle Routing Problems. In order to understand the differences between this heuristic and other heuristics that use large neighborhoods, an overview of the most related minima-escaping heuristics is provided in this thesis. Unfortunately, studies that involve real-world data are limited. Therefore, this thesis involves a study on how to configure the components of Adaptive Large Neighborhood Search, such that it can be used on real-world data. This study is carried out at ORTEC, a company that provides optimization software and analytical solutions to all sorts of companies, and customer cases are used for testing. Most of the time, an initial solution is required as input for the minima-escaping heuristics and the importance of the quality of this solution is not specified. Therefore, the influence of the construction method for an initial feasible solution on the performance of Adaptive Large Neighborhood Search will be investigated as well. It turns out that this influence is case and configuration specific. To intensify the search and explore smaller neighborhoods as well, local search methods are added to Adaptive Large Neighborhood Search, which will improve the solutions that are found. The research concludes with an Adaptive Large Neighborhood Search that is tuned for the real-world cases that are investigated at ORTEC. Because the computing time that is available for real-world cases is usually limited, a simplified version of the heuristic will be provided as well.

List of Abbreviations

ALNS	Adaptive Large Neighborhood Search
CI	Cheapest Insertion
CRR	Cluster Random Removal
CVRP	Capacitated Vehicle Routing Problem
CVRS	COMTEC Vehicle Routing Probem
CWR	Cluster Worst Removal
FSM	Fleet Size and Mix
HVRP	Heterogeneous Vehicle Routing Problem
KPI	Key Performance Indicator
LNS	Large Neighborhood Search
MDVRP	Multi-Depot Vehicle Routing Problem
MVRPB	Mixed Vehicle Routing Problem with Backhauls
ORD	ORTEC Routing and Dispatch
ORTEC	Operations Research Technology
PDP	Pickup and Delivery Problem
PCI	Parallel Cheapest Insertion
PRI	Parallel Regret Insertion
PI	Parallel Insertion
R&R	Ruin and Recreate
RR	Random Removal
SR	Related Removal (Shaw Removal)
TSP	Traveling-Salesman Problem
TVRP	Time-Dependent Vehicle Routing Problem
VNS	Variable Neighborhood Search
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPDP	Vehicle Routing Problem with Deliveries and Pickups
VRPDDP	Vehicle Routing Problem with Divisible Deliveries and Pickups
VRPMDP	Vehicle Routing Problem with Mixed Deliveries and Pickups
VRPO	Vehicle Routing Problem ORTEC
VRPSDP	Vehicle Routing Problem with Simultaneous Deliveries and Pickups
VRPTW	Vehicle Routing Problem with Time Windows
WR	Worst Removal
XML	Extensible Markup Language

Contents

Chapter 1

Introduction

Suppose you are the owner of a supermarket. Each day, you want your customers to be able to choose from a rich assortment of products. It is not acceptable to let your customers leave without all the products they wanted to purchase, because they were not in stock. Therefore, it is necessary to have a good supply. It would be best to have new products arriving within a certain part of the day, for example when the expected number of customers is low. However, the supply should not arrive hours after closing time, because your employees would have to wait for the supply to arrive in order to continue their work. Therefore, you would want the supply to arrive within a certain time window.

Most supermarkets are part of a supermarket chain. Therefore, there is a whole group of supermarket owners, who all want the best for their specific location. In case supermarkets are located in the same area, it would be beneficial to combine the supply of these supermarkets when looking for example at the costs for the driving time. However, one has to take into account several aspects, such as the time windows of these supermarkets, the driving time between them, and the capacity of the vehicle that is used. If, for example, supermarkets A and B need the supply to be delivered between 6 pm and 7 pm, but the driving time between the two supermarkets is more than an hour, these supplies cannot be combined in one vehicle. Besides the time window restrictions it may be impossible to deliver the supply of several supermarkets with one vehicle, because the combined amount of products will not fit in the vehicle. These aspects are just a small set of all restrictions that come along with a planning to supply all supermarkets of a supermarket chain. This raises the question on how to combine the supply to supermarkets, in such a way that there are no violations and all the demands are met.

This is an example of an optimization problem where routes need to be constructed that service the customer locations. Because all products are delivered at these customer locations, such as the supermarkets, this is a *distribution planning* of transports on routes. ORTEC B.V. is a company that works with these kind of problems. An introduction of this company is given in the next section.

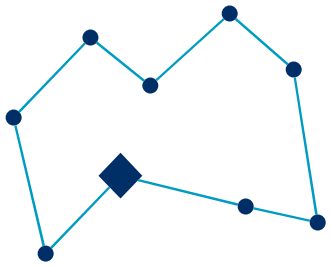
ORTEC B.V.

ORTEC B.V. is short for Operations Research Technology, in this report referred to as ORTEC. They provide optimization software and analytical solutions to all sorts of companies, such as carriers for supermarkets. It has grown from a small business, founded by Econometric students at Erasmus University Rotterdam in 1981, to a globally respected company. ORTEC is, for example, able to provide a good planning for which supermarkets need to be combined on a route for supply. Therefore, it is a perfect place to see in what way the theoretical heuristics that are discussed during the master Applied Mathematics, at the Delft University of Technology, are used in practice. In this thesis, data from customers of ORTEC will be used. Therefore, there are many restrictions, such as a limited computing time for finding a solution of good quality. Usually, real-world problems have more

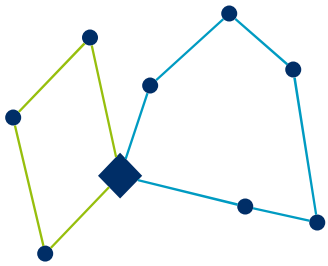
restrictions than the ones that are considered in the literature. Therefore, configurations of heuristics that are tested in the literature, may not provide solutions of good quality for the data that is used at ORTEC. More information about ORTEC and its products is given in Chapter 5.

1.1 Routing Problems and Algorithm Analysis

The problem that is described in the introduction of this chapter, is part of a class of problems called *linear integer problems*. More details on this class will be given in Chapter 3. An example of an optimization problem that is in this class, is the *Traveling Salesman Problem* (TSP). In a TSP, a network of nodes and edges is considered. Through all these nodes, one shortest circuit needs to be determined that ends and starts at the same location, and that visits all nodes exactly once. Because a shortest circuit is needed, the TSP is a *minimization problem*. One could think of the nodes representing the supermarkets form the introduction of this chapter and the start and end location as a depot where all the products are stored. An example is given in Figure 1.1a, where the diamond-shaped figure is the start and end point of the route. In this example, the nodes can be seen as supermarkets and the diamond-shaped figure as the depot where all the products for the supermarkets are stored, such as in the problem described in the introduction of this chapter. The reason for mentioning this TSP, is that it is strongly related to another optimization problem, called the *Vehicle Routing Problem* (VRP), which is one of the most studied problems in optimization problems regarding plannings for visiting locations. However, this problem was first proposed by Dantzig and Ramser (1959) under the name *Truck Dispatching Problem*. Instead of constructing one tour in the TSP, for the VRP multiple routes can be constructed. Because this problem is widely known under the name 'Vehicle Routing Problem', this report will refer to the problem with VRP as well. In this problem, the nodes or locations in the network have known demands and routes need to be determined such that the costs are minimal, they start and end at the same location, and the demands on the nodes are met. Figure 1.1b visualizes an example of a VRP. The supermarkets are supplied by two vehicles that both drive a circuit, or one vehicle that drives two circuits. The main difference between the TSP and VRP are the demands and possible multiple routes in the VRP. The



(a) An example of a TSP, where all nodes in the network are visited exactly ones, within one circuit.



(b) A solution to a VRP, including two routes that together visit all customers in the network that have transportation request for the diamond-shaped depot. Both routes start and end at the depot, which results in two circuits.

Figure 1.1: Two examples are given: on the left a TSP, on the right a VRP.

VRP is part of a class of problems: *linear integer programming problems*. One can choose to relax the problem into a *linear programming problem*. Solving this relaxation does not imply that the original problem is solved as well. More about this is included in Chapter 3. There is an algorithm for linear programming that will find an optimal solution in a number of steps. This number of steps is known to grow as a polynomial in the size of the problem instance that is looked into (Papadimitriou & Steiglitz, 1998). This size is measured as an integer n. The kind of algorithms are called *polynomial algorithms* or *polynomial-time algorithms* and are known for their not too fast increasing running time as the size of the

problem increases (Cook, Cunningham, Pulleyblank, & Schrijver, 1998). Note that the TSP and VRP do not belong to the class of linear problems, and hence cannot be solved with such an algorithm. The running time of these algorithms is presented as a polynomial of the form

$$a \cdot n^k + \mathcal{O}\left(n^l\right) = \mathcal{O}\left(n^k\right),$$

where $a, n, k, l \in \mathbb{N}$ and $l < k$. However, for large k, the polynomial algorithms may have a long running time. All decision problems that can be solved in polynomial time are in the complexity class that is denoted by $\mathcal{P}$. There is another class of decision problems, which is the $\mathcal{NP}$ class. This class contains all decision problems for which is known that they can be verified in polynomial time. These problems are stated as questions, for which can be verified, in polynomial time, that the answer to these questions is positive. For decision problems that are in the $\mathcal{NP}$ complexity class, it is not necessary that the solution for these problems can be found in polynomial time. From this definition follows that $\mathcal{P} \subseteq \mathcal{NP}$. Whether $\mathcal{P} = \mathcal{NP}$ is yet unknown and providing a proof whether this is indeed the case is one of the well-known 'Millennium Problems'. Furthermore, the class $\mathcal{NP}$ contains another class: $\mathcal{NP} - complete$ class. A problem is in this class in case all other problems that are in $\mathcal{NP}$ can be polynomially transformed to this problem. For more information on this subject we refer to Garey and Johnson (1979), Papadimitriou and Steiglitz (1998) and Arora and Barak (2009).

1.2 Problem Description and Motivation

When working with a more practical VRP using for example multiple depots, multiple trips and time windows, the problem will be harder to solve than the original VRP that is described in the previous section. A more complex VRP is called *rich* and heuristics need to be used to find feasible solutions. Unfortunately, using heuristics will not guarantee finding the optimal solution, but one can get very close to the optimum. Therefore, the quality of the solution needs to be examined. There is always a trade-off between the quality of the objective function value of a solution and the computing time. A short computing time and high quality solution are desirable, but achieving both at the same time can be difficult. Usually, when working with heuristics that are iterative, spending more time on calculations will provide a better solution (Acharya, 2013). However, the available time for calculations is limited in practical cases, so the quality of the solution is also typically limited. Another difficulty that comes along with heuristics, besides this trade-off, is that they are case-specific (Acharya, 2013). The behavior and effectiveness of heuristics depend on the problem that is looked into. For example, a heuristic that works well for a VRP with 10 locations and time window restrictions may not produce a good solution for a VRP with 200 locations without time window restrictions. Therefore, heuristics should always be tested on the problem that is looked into, especially when you are working with real-world problems. This is mainly due to all the restrictions that are most of the time problem specific for such real-world problems.

1.2.1 Large Neighborhoods

Most heuristics consist of two phases, sometimes expanded with more steps. The first phase is called the construction phase and consists of the construction of a feasible initial solution. This initial solution will be improved step by step during the improvement phase (Bräysy, 2003). During this phase the solution is iteratively modified with the help of local search steps. Little alternations in the solution can already provide a new feasible solution that is better than the initial solution. The main idea is to keep searching for new feasible solutions in the neighborhood of the current solution, while improvements are found (Papadimitriou & Steiglitz, 1998). A *neighborhood* of a solution s is usually defined as all the solutions that lay within a certain area of s. To be more precise, the neighborhood of s is defined as

$$N_\epsilon(s) = \left\{ \|y - s\| \leq \epsilon, \text{ given that } y \text{ is a feasible solution} \right\}.$$

With the help of local search, these solutions $y \in N_\epsilon(s)$ can be found. Usually, local search includes small changes in the construction of the solution, such as interchanging route segments. This usually leads to small changes in the value of the objective function (Schneider & Kirkpatrick, 2006, p. 73). Hence, small ϵ are required. In this way, when the search reaches a local minimum, it can be hard to find a better solution during next iterations. This idea is visualized in Figure 4.1. Especially when complex problems are considered with many constraints, altering some parts of the solution may produce an infeasible solution (Schrimpf, Schneider, Stamm-Wilbrandt, & Dueck, 2000).

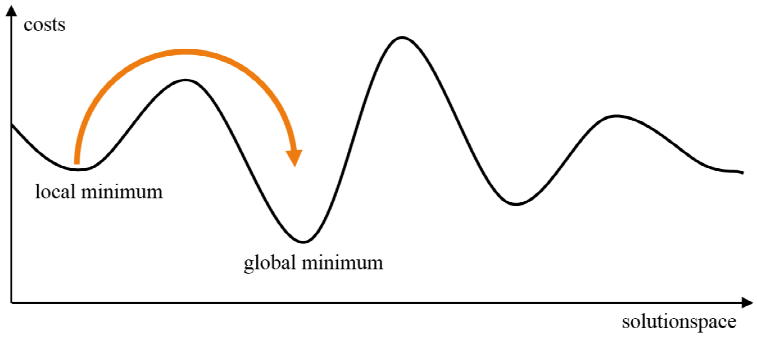


Figure 1.2: A figure to illustrate the principle of escaping a local minimum in the search for a global minimum. Note that this figure is an example of varying one descision variable that influences the costs. The costs in this case can be seen as the objective.

Besides the complexity of finding a new minimum, the quality of a local minimum is in most cases not good enough, especially when the difference between this local minimum and the global minimum is large. Therefore, it can be useful to search through larger neighborhoods. This might require having a higher value for ϵ, to escape local minima. In our case, we define a *large neighborhood* of a solution s as solutions that can be found when altering a large part of s. For example, by interchanging many segments of many routes at once.

When investigating larger neighborhoods in the search for a better local minimum, one can escape local minima. There are methods that are able to work with both small and big changes in a solution, and thus use small and large neighborhoods. They are related to methods called *Ruin and Recreate* (R&R) and *Large Neighborhood Search* (LNS), that both rely on the same idea of using large neighborhoods to investigate the solution space. In each iteration of these heuristics the current solution is partially 'ruined' or 'removed' and will be 'recreated' to find a better solution. Another method that uses this strategy is called *Adaptive Large Neighborhood Search* (ALNS). This method is based on R&R and LNS, with some additions for better performance. In the literature this seems to be a promising method of finding solutions for complex and case-specific VRPs. To ruin solutions, one has to choose a 'removal method' to unplan a predefined number of transportation requests from the current solution. This will lead to a partial solution, where the unplanned transportation request needs to be inserted again, hopefully at a better position than in the previously found solution. The reinsertion is done by an insertion method that is chosen from several 'recreate methods'. For more information is referred to Golden, Raghavan, and Wasil (2008).

1.3 Research description

As already described, a rich VRP is hard to solve. Therefore, smart heuristics are needed to find near-optimal solutions. Large Neighborhood Search, and variants of this heuristic, are known to find these near-optimal solutions. Therefore, the software of ORTEC incorporates

Large Neighborhood Search as heuristic to improve the solutions that are constructed by their software as well. To obtain more insight into how to configure the algorithm and how to configure all of its components, this research is done. Unfortunately, most of the test cases from the literature involve a VRP that is not rich. Therefore, this research also includes tests to see whether the conclusions from the literature about Large Neighborhood Search also apply to rich cases from ORTEC customers.

The heuristic that will be used during our tests will be based on Adaptive Large Neighborhood Search (ALNS), introduced by Ropke and Pisinger (2006b), that involves removing and reinserting transports from a solution to escape local minima. They use several *removal methods* and several *recreate methods* within their algorithm, that are chosen with the help of a *roulette wheel*. More details about their ALNS are included in Chapter 4. Our contribution to this method are tests on the influence of the construction method for the initial solution on ALNS, as Ropke and Pisinger take a feasible initial solution as input for their heuristic. Furthermore, local search is incorporated in the heuristic as well, which is not done by Ropke and Pisinger to the best of our knowledge. Because we will execute our tests with software from ORTEC, the configuration of the components of the heuristic are different as well and tests may produce different findings on our real-world cases than the findings from the literature.

Furthermore, as already explained in Section 1.1, the computing time for a heuristic is very important. Because of a wide range of possibilities within heuristics such as Large Neighborhood Search, the computing time can increase very fast, especially for a rich VRP. For example, it is not acceptable for a supply planning for supermarkets to take 20 hours to create a planning. Most of the time, each day a new planning needs to be created, and hence a computing time of less than an hour is more appreciated. However, there is not one universal limit on the computing time for rich VRPs, this depends on the instance that is looked into. Therefore, we must add a limited possible computing time, depending on the instance that is considered, to the goal of using Adaptive Large Neighborhood Search successfully on a rich VRP. Therefore, the main question for this research is the following.

In what way can the Adaptive Large Neighborhood Search heuristic be configured to find near-optimal solutions for rich and real-world VRPs, such that the computing time needed is acceptable for the instance that is looked into?

Note that the main goal is to gain information on how to configure the different components of the ALNS, such that the combination of the components produces the best solutions. 'The best solutions' is defined as the solution that is ranked first in case the solutions that are found during our research are ordered according to the KPIs and computing time. In order to answer the main research question, we state subquestions that will be answered during our research.

- Which construction method is needed to build an initial feasible solution, such that Adaptive Large Neighborhood Search performs the best?
- Which removal methods and corresponding settings are needed, such that Adaptive Large Neighborhood Search performs the best?
- Which recreate methods and corresponding settings are needed, such that Adaptive Large Neighborhood Search performs the best?
- When and where are local search steps needed in the Adaptive Large Neighborhood Search heuristic, such that Adaptive Large Neighborhood Search performs the best?

In these subquestions 'performs the best' asks for a definition. As an example we take the first subquestion, regarding the construction method for the initial solution. We will test several construction methods followed by a version of ALNS to improve the initial solution. According to the value of the objective function for the solutions that are found, the methods are ranked. The method with the best value for the objective function will be defined as the 'best performing construction method' in combination with ALNS. However, as already explained, the computing time plays a role in this ranking as well. We will need

to weigh the performance regarding the value for the objective function and the computing time.

The subquestions are used as a tool to get more insight into the usage and influence of the construction method for the initial solution, removal and recreate methods and local search steps. For example a construction method that is very complex might produce a good solution. However, when for example an intensive local search is used, a complex construction of the initial solution may not be necessary.

1.4 Outline

To provide more information about the VRP, Chapter 2 contains details on the problem. Many variants of the VRP exist due to all kinds of restrictions that can be included, such as limited capacity for the vehicles or time window restrictions on the product deliveries. The variants of the VRP that are relevant for ORTEC are described in Chapter 2 as well. This includes the Capacitated VRP, Heterogeneous VRP, VRP with Backhauls, VRP with Time Windows, Multi-Depot VRP, Time-Dependent VRP, Pickup and Delivery Problem and the VRP with Deliveries and Pickups. The Chapter concludes with the restrictions that are present in all customer instances from ORTEC that are used for testing.

The VRP that is used for ORTEC is a rich VRP with real-world restrictions. These restrictions are hard to formulate in a way such that the model can be solved to optimality. Information on how to model a VRP is described in Chapter 3. Multiple objectives are used at ORTEC to formulate the objective function, which is included in this chapter as well. The difficulty of formulating the restrictions into a model is described and the choice for not constructing an explicit mathematical model is motivated as well.

The first three chapters are needed to understand the difficulty of testing on real-world instances. Heuristics that may be used to find near-optimal solutions for VRPs involving these instances are given in Chapter 4, including metaheuristics that belong to the local search class: Tabu Search, Iterated Local Search, Variable Neighborhood Search, Large Neighborhood Search and Ruin and Recreate. These metaheuristics are needed to place the heuristic, that is used for testing, in the literature. The heuristic that is used for testing is based on Adaptive Large Neighborhood Search, which is described in Chapter 4 as well.

More information about the software that is used for this research is included in Chapter 5. We elaborate on the input for this software, in what way the configurations are provided as input (explained in detail with an example) and how to interpret the output.

After it is described in what way the ORTEC Adaptive Large Neighborhood Search is related to the metaheuristics that are known in the literature, the components of this heuristic are reviewed in Chapter 6. Per component a short introduction is given, followed by definitions and the performance that is given in the literature. Four components are included: the construction method for the initial solution, the removal methods, the recreate methods and the acceptance criterion for accepting new found solutions during the search. This literature review will provide a starting point for tests that are reported on in this thesis.

Because many components are tested that are combined into one heuristic, details on the testing approach are provided before the experiments are described and reported on. The testing steps are described and motivated and information about in what way the solution quality is measured is included as well. The chapter concludes with an overview and description of the data that is used for our experiments.

The results from the tests that are executed for this research, are reported on in Chapter 8. The chapter starts with the planning that is made with the original customer configurations, in Section 8.1. Initial tests are performed and reported in Section 8.2 to have initial frameworks for the hybrid method that will be tuned. This tuning will be done per component of the hybrid method and the results are shown in Sections 8.3 up till 8.5. After the tuning of the components, two configurations will be constructed in Section 8.6. One configuration will be constructed with the components that resulted in the lowest KPIs, and the other configuration will be recommended in case a simple heuristic is required that

uses only one removal and one recreate method.

The subquestions and main research question will be answered in the conclusion in Chapter 9. Recommendations for further research are included in as well, together with recommendations for ORTEC.

Chapter 2

VRP Classification

From the introduction it is clear that this thesis considers finding a close-to-optimal solution for real-world VRPs. However, as already explained, there are many restrictions that may or may not be taken into account. These restrictions cause the class of VRPs to be divided into sub-classes, that are not mutually exclusive. This is due to the fact that one class of VRPs may or may not include some of the characteristics of another class of VRPs. To give an idea of these classifications, an overview of the most relevant VRPs is given in this chapter. They all have some properties that are included in VRPs from customers at ORTEC. However, this list is not complete, but it gives an idea of the magnitude of different restrictions and properties ORTEC is dealing with.

This chapter starts with the original VRP, in Section 2.1, and we define some terminology to prevent misunderstandings. Next, an overview is given of eight different classes: the capacitated vehicle routing problem, Section 2.2, the heterogeneous vehicle routing problem, Section 2.3, the vehicle routing problem with backhauls, Section 2.4, the vehicle routing problem with time windows, Section 2.5, several multi-depot vehicle routing problems, Section 2.6, the time-dependent vehicle routing problem, Section 2.7, the pickup and delivery problem, Section 2.8 and the vehicle routing problem with deliveries and pickups 2.9. To conclude this chapter, we define a VRP that is used for this thesis in Section 2.10.

2.1 Vehicle Routing Problem

First of all, we recall that the goal for a VRP is to determine a set of routes to serve all transportation requests at a minimum costs, so the VRP is a minimization problem. The requests contain a transportation from a central depot to a location in the network. This network consists of a depot, customer locations (nodes) and links between these locations and the depot. For the routes there are two conditions: they must start and end at the depot and all locations in the network must be visited exactly once. Recall that in Figure 1.1b an example of a VRP with two routes is given.

Terminology

There are some terms that need extra explanation. In the literature a lot of terminology is used for the same objects, events and procedures. To avoid misunderstandings, the terminology that is used in this thesis will be given.

All different locations in the network need to be specified. There are *customers*, *depots* and *home addresses* for the drivers of the vehicles. It may happen that the distance or driving time for a truck driver from his home address to the depot is taken into account for the minimization problem as well. However, most of the time, this is not included, and we assume that the driver is already located at the depot. In the VRP, there are transportation requests with goods that need to be delivered or picked up. For these transportation requests often different terms are used. Solomon (1987) is talking about delivery of goods to customers, Toth and Vigo (2014) have transportation tasks, El-Sherbeny (2010) service

customers, Pisinger and Ropke (2007) just talk about requests and sometimes the term orders are used. In this thesis, the transportation request are called *transports* and vehicles serve customers. The transports always have a *pickup location* and *deliver location*, where most of the time the pickup location is a depot and the deliver location is at a customer. These visits at locations are called *tasks*. Hence, a transport consists of two tasks: a *pickup task* and a *deliver task*. There may be other tasks for the driver as well, such as loading and unloading of goods, waiting to be able to deliver or pick up goods or resting periods. Furthermore, although the problem is called a vehicle routing problem, there are multiple terms used for the routes that are constructed. Shaw (1997) refers to the routes with both vehicle routes and tours, as they have the same start and end point (a depot), whereas Pisinger and Ropke (2007) call them routes. Within this thesis report we use *routes*, where a route starts and ends at the depot. However, in Chapter 8 we report on the experiments that are done for this thesis and the routes are called *trips*. Within the software of ORTEC it is possible to cut the routes into parts and assign these parts to different vehicles. These smaller parts are called trips in the software, and a route may consist of several trips. However, in this thesis, all routes consist of one trip and hence they are the same.

2.2 Capacitated Vehicle Routing Problem

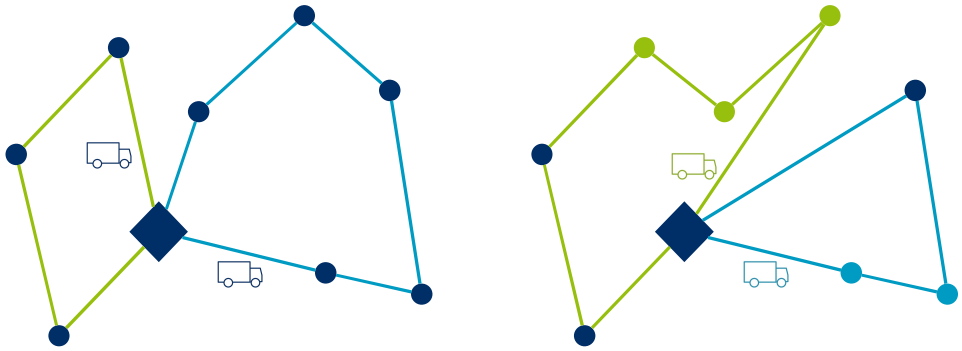
The *Capacitated Vehicle Routing Problem* (CVRP) is the variant of the VRP that is most studied in the literature. In this CVRP customers have to be served using vehicles that have limited capacity, which is the extra restriction compared to the original VRP. The demand of the customers is, just as in the VRP, known. The fleet of vehicles is *homogeneous*, therefore all vehicles are identical and have the same capacity. Just like the VRP, the CVRP is a distribution problem and transports need to be planned from a depot to customers in a network. An illustration of this CVRP is given in Figure 2.1a.

2.3 Heterogeneous Vehicle Routing Problem

So far, only a homogeneous fleet has been considered, but a VRP can also have a *heterogeneous fleet*. In the *Heterogeneous Vehicle Routing Problem* (HVRP), there are different types of vehicles available in the network. This variant of the VRP has many versions, because 'different types of vehicles' can be interpreted in several ways. The vehicle can have different capacity limitations, (fixed) costs, travel times, time windows, serving areas and other specifications. One could also have a limited or unlimited number of vehicles that can be used. Usually, the HVRP includes a limited number of vehicles, and *Fleet Size and Mix* (FSM) refers to the problem where an unlimited number of vehicles is available (Golden et al., 2008). Figure 2.1b gives an example of a HVRP, which may be compared to the CVRP in Figure 2.1a.

2.4 Vehicle Routing Problem with Backhauls

A logical extension of the VRP is the *Vehicle Routing Problem with Backhauls* (VRPB). Besides distribution also collection is possible in this variant. For example, when products are delivered in trays, these trays need to be transported back to the depot. This results in transports with a pickup action at a customer location and a deliver action at the depot. Figure 2.2 gives two examples of the extended VRP from Figure 1.1b, including the backhaul customers, so the customers that require a pickup. First, consider Figure 2.2a. Deliveries from the depot go to *linehaul customers*, the dark blue nodes, and all collections that have to go back to the depot are transported from *backhaul customers*, the orange nodes. The vehicle needs to be empty, so all transports are done, before visiting a backhaul customer. Hence, delivery and pickup goods can not be transported at the same time. When this last restriction is left out of scope, backhaul customers may be served before all linehaul customers on the route are visited. In this case the problem is called a *Mixed Vehicle*

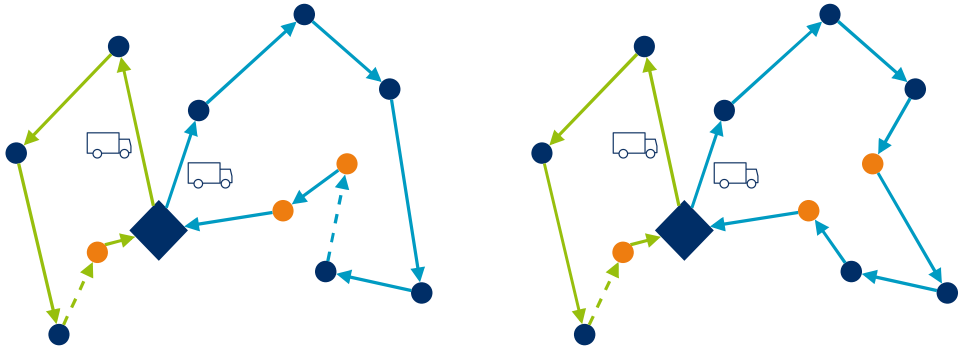


(a) A solution to a CVRP, including two routes and two vehicles. Note that the only difference between the VRP and CVRP is that the vehicles have a limited capacity. It may happen that the routes in the VRP exceed the capacity of the vehicles, so it may happen that the optimal solution for the CVRP is different than for the VRP.

(b) Two different types of vehicles, that both have their own route. The green customers can only be served by the green truck, the light blue customers only by the light blue truck, and the dark blue customers may be served by both the green and the blue truck.

Figure 2.1: Two different ways of extending the VRP: add capacity constraints to the vehicles (a), resulting in a CVRP, and having different types of vehicles (b), a HVRP, coming with different restrictions and constraints.

Routing Problem with Backhauls (MVRPB), see Figure 2.2b. Note that in Figure 2.2 the connections between nodes have direction now, because all linehaul customers need to be visited before all backhaul customers. From now on, backhaul customers will be referred to as *pickup customers*, because products are picked up at the customer location, and linehaul customers as *delivery customers*, because products are delivered at the customer location.



(a) An example of a VRPB, where pickup customers (in orange) are included in the network. Note that direction on the edges is included, and thus the routes have a direction as well. A dashed link between nodes in the network represents that the vehicle is empty. The vehicle is empty after the last delivery customer in the route and before the first pickup customer is visited.

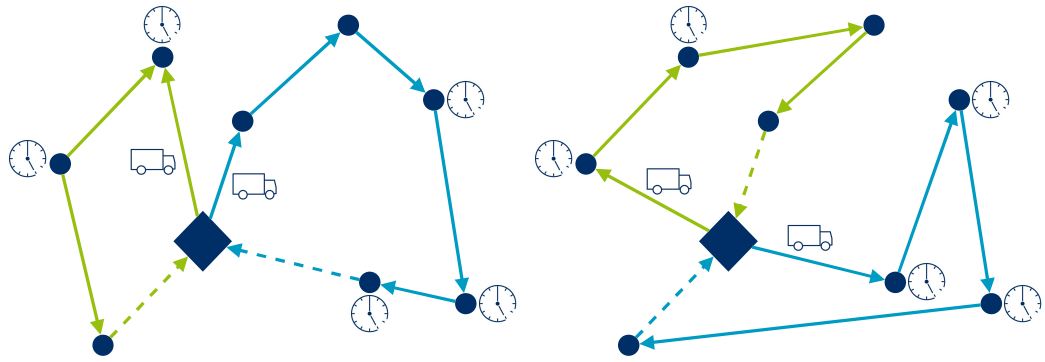
(b) Another example of a VRPB, where it is allowed to pickup transports from pickup customers before all delivery customers are visited. This is known as a MVRPB.

Figure 2.2: Two examples of solution for a VRPB, the general version (a) and an extension: the MVRPB (b). In the extended version it is allowed to pickup goods before the vehicle is empty.

2.5 Vehicle Routing Problem with Time Windows

Another extension of the VRP, is the *Vehicle Routing Problem with Time Windows*. This problem includes *time windows* at the customer locations. The problem can include *hard time windows* or *soft time windows*. The vehicles must visit the customers within the given time interval, when the time windows are hard. On the other hand, when soft time windows are used, the vehicle may visit the customer outside the time interval. In this case it is common that vehicles can not arrive after the end of the interval, but they may arrive before the start of the interval. The latter will usually cause a penalty in the objective function or some wait time is added to make the planning feasible. An example of such a VRPTW is given in Figure 2.3a. Note that not all customer locations necessarily need restrictions on their time windows. It may happen that some customers have time windows that are non-restrictive, such as soft time windows. However, this is not very common in real-life, as locations usually have opening hours. Because there are no pickup customers in this example, the vehicles return empty at the depot, which is visualized as a dashed line.

It may happen that the time windows are very *tight*. That means that the start and end time of the time windows are very close, so there is only a short amount of time for the transports to be delivered. It may happen, because of these tight time windows, that the vehicles are forced to visit the customers in a specific order. This could increase the value of the objective function a lot, for example when the quality is measured in distance. This is illustrated in Figure 2.3b. Consider the blue route, where it would have been beneficial to plan the first stop after the third stop. This would decrease the distance that needs to be driven. However, due to the time window of the first stop, this is not possible.



(a) An example of a VRPTW, where the locations in the network with time window restriction are visualized with a clock. Note that not all locations have time window restrictions.

(b) Another example of a VRPTW. The time windows in this case are very tight, which causes the routes to be different than in the figure on the left.

Figure 2.3: Two examples on how different time windows will influence the planning in the VRPTW.

2.6 Multi-Depot Vehicle Routing Problem

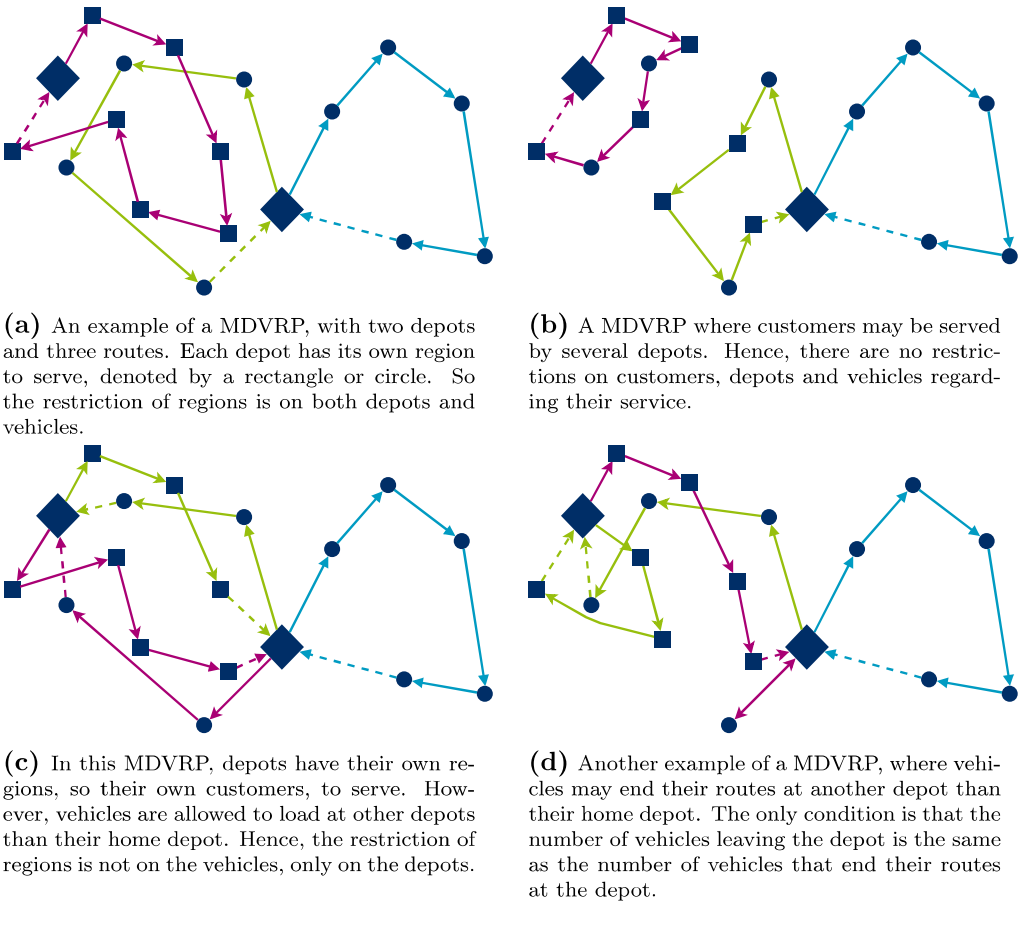
In case there is not one depot in the network but several depots, the VRP turns into a *Multi-Depot Vehicle Routing Problem* (MDVRP). There exist several versions of this MDVRP, so 'multi-depot' may be explained in multiple ways.

- Customers can be served by several depots or each depot serves its own region. A region can for example be based on zipcodes, or certain customers that only can be served by a certain depot. In the latter case, it may happen that these customers are scattered all over the network. A consequence would be the overlap of regions between different depots. This version of the MDVRP is visualized in Figures 2.4a

and 2.4b, where rectangular locations belong to the region of the depot on the left, and the circular locations belong to the other depot. In case each depot has its own region to serve, see Figure 2.4a, the MDVRP can be seen as a combination of several VRPs. It may be solved by splitting the problem into as many VRPs as there are depots, because the MDVRP can be solved per region. On the other hand, when customers may be served by several depots, see Figure 2.4b, the problem can not be split up into several VRPs. In this case, customers may have a preferred depot, but they may be served by other depots. One can, for example, include penalties in case customers are not served by their preferred depot.

- Vehicles may visit several depots. Note that the customers still need to be served by a specific depot, which makes it a different version from the first item. Figure 2.4c shows an example of this. Vehicles may start their routes at their home depot, transport goods to the customers, visit another depot to load goods and transport these goods to other customers. Finally, they return to their home depot.

- The number of vehicles that arrive at a depot must be the same as the number that leave the depot. Note that in this case the vehicles may end up at a different depot than where they started at, which makes it different from the second case. An example is given in Figure 2.4d.



(a) An example of a MDVRP, with two depots and three routes. Each depot has its own region to serve, denoted by a rectangle or circle. So the restriction of regions is on both depots and vehicles.

(b) A MDVRP where customers may be served by several depots. Hence, there are no restrictions on customers, depots and vehicles regarding their service.

(c) In this MDVRP, depots have their own regions, so their own customers, to serve. However, vehicles are allowed to load at other depots than their home depot. Hence, the restriction of regions is not on the vehicles, only on the depots.

(d) Another example of a MDVRP, where vehicles may end their routes at another depot than their home depot. The only condition is that the number of vehicles leaving the depot is the same as the number of vehicles that end their routes at the depot.

Figure 2.4: Examples of MDVRP, all having different conditions and restrictions for both the depot, the customers and the vehicles.

2.7 Time-Dependent Vehicle Routing Problem

In the original VRP the distance between two customers is used to compute the travel time. For the *Time-Dependent Vehicle Routing Problem* (TVRP) the time of day is also taken into account. For example, when a vehicle is driving to a location it will take more time during rush hour than during night hours. Therefore, congestion needs to be taken into account, which makes the problem more realistic. When working with congestion, predictions need to be made. There are all sort of ways to include congestion. This could be done by simply adding some extra time to the travel time during rush hour, or for example using more complicated forecasting models and methods. Note that congestion does not only depend on the time of day, but also on the locations and roads. Congestion on roads in small towns has different behavior than congestion in main streets. Therefore, the road network can have an influence on congestion as well.

2.8 Pickup and Delivery Problem

The *Pickup and Delivery Problem* (PDP) can be seen as a generalization of the VRPB, where transports have even more possibilities. In the PDP transports are not only allowed to go from or to a depot, but they also may be picked up from a customer and be delivered at another customer (Sol & Savelsbergh, 1992). The planning of transports on routes will be different than the planning in the VRP, because in the PDP vehicles may load goods at customer locations, instead of only at the depot. Within the PDP, routes may have more transports on average, because vehicles are now able to load after visiting a few customers. In the PDP deliveries must be done before pickups. Note that this class of problems is not the same as the next extension of the VRP (Nagy, Wassan, Speranza, & Archetti, 2015).

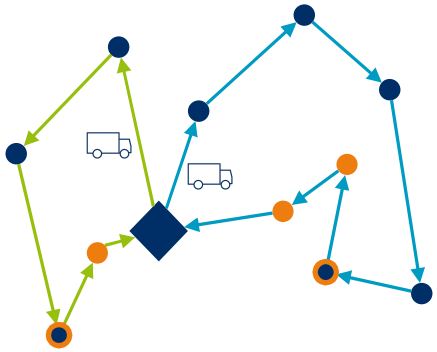
In the literature, VRP, VRPB and PDP are categorized in a different way as well. The categories divide the nodes in the network into 'many'-nodes and 'one'-nodes, where 'many' represent the customers and 'one' refers to the depots. These categories for the problems are *many-to-many* (PDP), *one-to-one* (from depot to depot), *many-to-one* (collection) and *one-to-many* (distribution) problems.

2.9 Vehicle Routing Problem with Deliveries and Pick-ups

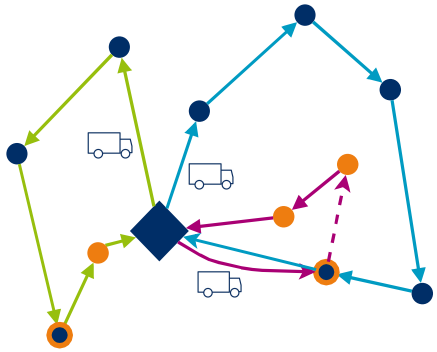
In the *Vehicle Routing Problem with Deliveries and Pickups* (VRPDP) customers may have transports to be delivered and to be picked up (Nagy et al., 2015). In this VRPDP a customer can either be a pickup or delivery customer, or both. Note that for the VRPDP it is not allowed to have goods being transported between customers directly, therefore it is not a PDP. What makes this VRPDP special, and different from the VRPB, is that customers may be visited twice when they have both a pickup and delivery transport. The VRPDP has a few sub classes, which are visualized in Figure 2.5.

- In case all customers are either pickup or delivery customers, they fall into the class of distribution or collection, and not both. Therefore, this version of the VRPDP becomes a VRPB.
- When a customer has both a pickup and delivery demand and only one stop is allowed, the VRPDP is called a *Vehicle Routing Problem with Simultaneous Deliveries and Pickups* (VRPSDP). An example is given in Figure 2.5a. In case the pickup demand is much larger than the delivery demand, this problem may become a very hard planning problem. In this way it will be hard to plan transports on routes, because of the vehicle capacity. There is only one stop allowed per customer, so the planning needs to take into account the volume of the pickup transport, in order to be able to visit the customer for the delivery goods.

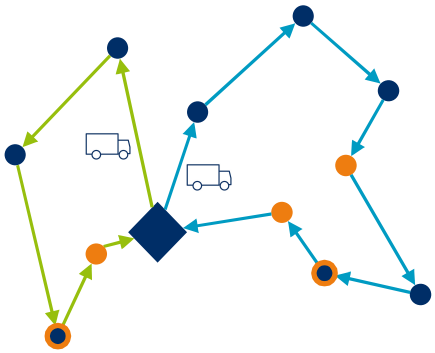
- For the *Vehicle Routing Problem with Divisible Deliveries and Pickups* (VRPDDP), a customer with both a pickup and delivery transport may be visited twice, visualized in Figure 2.5b. Note that in case a customer only has a pickup (or delivery) transport, it is not allowed to visit this customer twice.
- The last class is the *Vehicle Routing Problem with Mixed Deliveries and Pickups* (VRPMDP), where pickups and deliveries may be done in any order on a route. Therefore, it is not necessary to deliver all transports before pickups are done, in contrast to most other problems that are described in this thesis. An example is given in Figure 2.5c.



(a) An example of a VRPSDP, where two customers in the network are both pickup and delivery customers. This is denoted with a blue and orange circle. These customers are only allowed to be visited once, and all deliveries have to be done before the first pickup is possible.



(b) This figure gives an example of a VRPDDP. In this problem, it is allowed to visit a customer twice, provided that it is both a pickup and deliver location. All goods need to be delivered before any goods can be picked up.



(c) In the VRPMDP, it is allowed to visit a pickup location before all deliveries are done in a route.

Figure 2.5: Three examples of different VRPDPs: VRPSDP (a), VRPDDP (b) and VRPMDP (c).

2.10 Vehicle Routing Problem ORTEC

In the previous sections many variants of the VRP are discussed. Most of them are relevant for the problem that is tested in this thesis. Multiple instances are used for testing, so we will not go into detail about specific restrictions or capabilities. However, all instances do have the following in common.

- All vehicles have limited capacity for all the instances that are used during the experiments. The number of vehicles that may be used is limited as well.

- There are different restrictions on the vehicles, which results in a heterogeneous fleet. The vehicles have different capacity limitations and different capabilities. For example, there are vehicles with and without tailboard, some vehicles may only be used for specific goods, they have different capacities and they are able to cool or not. The vehicle capabilities and restrictions are customer and case specific, hence we will not go into detail.
- The locations in the network all have time windows in which vehicles may arrive, related to opening hours, but also the transports have specific time windows in which they must be picked up and delivered. In order to plan a transport on a route, it is necessary that the opening hours for the customer locations, hence the customer time windows, overlap with the delivery time window specified for the transport. The time window restrictions on the customer locations are hard time windows, vehicles must enter the locations one minute before the end of the time window. The time window restrictions are also hard for the transports that need to be delivered. In some cases appointments need to be made at the customer location, to reserve a dock for unloading. These appointments cause tight time windows on the transports.
- All instances that are used for testing have one depot.
- The transports that need to be planned on routes all need to be picked up at a depot and delivered at a customer location in the network. Therefore, all instances belong to the distribution class, no collection at the customers takes place.
- The customers of ORTEC that use the planning software take into account congestion. Hence, for the software that the customers of ORTEC use, congestion is configured. However, because we test on a computer from ORTEC, not at the actual computer that is located at the customer of ORTEC, congestion is not taken into account. Therefore, our experiments exclude congestion.

The combination of all restrictions result for us in the so called *Vehicle Routing Problem ORTEC* (VRPO), which is a complex and close to realistic problem. We refer to it as 'close to realistic', because not all complications from real-world planning are taken into account. For example, congestion is not included, so we do not have a time-dependent problem.

Chapter 3

Mathematical Model

In the previous chapter an overview was given of some of the variants of the VRP. To be able to solve these variants of the VRP, mathematical models need to be formulated. This can be done in many ways. However, when working with rich VRPs that have real-life restrictions and characteristics, it is very hard to formulate a mathematical model.

This chapter starts with an introduction on linear and nonlinear programming problems in Section 3.1. To include some information on restrictions of the model, and the difficulties of them for a rich VRP, and the usage of upper and lower bounds when a model is constructed, a short overview is given in Section 3.2. We also motivate our choice for not including a model as well. Nevertheless, we do use an objective function. The way the objective function is constructed for our VRPO is given in Section 3.3.

3.1 Models

A general form of a model for a programming problem, when the problem is to minimize, is given by

$$\begin{aligned} \underset{(\mathbf{x},\mathbf{y})}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \gtreqless \mathbf{b}, \\ & \mathbf{x} \in \mathbb{Z}^n, \mathbf{y} \in \mathbb{R}^p, \end{aligned}$$

where $\mathbf{c} \in \mathbb{Z}^n$, $\mathbf{d} \in \mathbb{R}^p$, $\mathbf{A} \in \mathbb{Z}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{m \times p}$ and $\mathbf{b} \in \mathbb{R}^m$ where $m = \max(n, p)$. In case the model only consist of linear functions, the model is called a *linear programming problem*, LP. In case the functions are general, not all linear, the problem is called a *nonlinear programming problem*. The function that includes $\mathbf{c}$ and $\mathbf{d}$ is called the *objective function*, which needs to be minimized with respect to the vectors $\mathbf{x}$ and $\mathbf{y}$ that consist of decision variables. For the LP, all feasible solutions lay within the polyhedron that is defined with the use of the linear inequalities. In case all decision variables are contained in $\mathbb{Z}$, and all functions are linear, the problem is called an *Integer Linear Problem*, ILP. Also a combination of continuous and integer variables is possible, which is given as the general form above, the problem is a *Mixed Integer Linear Problem*, MILP. All parameters in $\mathbf{A}$, $\mathbf{B}$, $\mathbf{c}$ and $\mathbf{d}$ may vary per problem that is looked into.

3.2 Restrictions and Upper and Lower Bounds

There are many models known in the literature that are used to solve all the variants of the VRP that are described in Chapter 2. For real-world cases, many constraints of all sorts of variants of the VRP need to be combined. There will be many constraints that influence each other, and hence it will be hard to combine all constraints in a model. Therefore, it is possible to relax the restrictions if necessary. However, if such a model is constructed, it might be hard to solve, even with software that is known for their good performance. In

this case, one can relax the whole model. For example, when a problem is modeled as an ILP, the restriction of having only integers can be relaxed. Then, the model becomes a LP. Solving this model will provide an optimal solution $\mathbf{x}_{LP}$, for which holds

$$f(\mathbf{x}_{LP}) \leq f(\mathbf{x}_{ILP}),$$

because the LP is a relaxation of the ILP and we consider a minimization problem. Moreover, any feasible solution $\mathbf{x}$ for the minimization problem can be used as an upper bound. Hence, this would conclude in

$$f(\mathbf{x}_{LP}) \leq f(\mathbf{x}_{ILP}) \leq f(\mathbf{x}).$$

However, with many relaxations, one can question the profitability of having these bounds. We can speculate that the gap between the upper and lower bound will be very large. Therefore, one would have to provide an indication of quality of the bounds, especially the lower bound as we are talking about minimization problems.

Constructing a model for real-world VRPs requires a lot of effort and could be a study on its own. Furthermore, we do not need a mathematical model of the VRPO to be able to use the software of ORTEC that will be used in our research. The goal of constructing such a model is to find feasible solutions that lay within a polyhedron. However, the focus of this research is rather on local search and different neighborhoods, that both do not require a model. Therefore, we do not put effort in constructing a model, because we will not use it in our research. Nevertheless, we do work with an objective function that has a special hierarchy, which will be described in the next section.

3.3 Objective Function

There are many ways to formulate the objective function of a VRP, for example one single objective can be minimized or multiple objectives can be combined in one function with weights. This can be done in a linear or non-linear fashion.

For ORTEC, the objective function works with different layers. Multiple objectives are used within the objective function, they are specified with an order of importance. Maximizing the number of planned transports is a common choice for the main objective. For example, usually the plan costs increase in case the number of planned transports increases. Therefore, when the total plan costs is taken as main objective, the software will not plan any transports, as the plan costs remain close to zero. Maximizing the number of planned transports is seen as a setting that can be configured. From now on, we can refer to our problem as a minimization problem. For our problem, the following objectives are taken into account during the optimization process:

1. **Plan costs.** The total plan costs, in euros, are minimized and usually contain the following three components:

   - costs per used route,
   - costs per hour that the route lasts,
   - costs per kilometer that is driven in the route.

   The plan costs are summed up over all routes and represent the total costs in the network. Each route, depending on the vehicle that is used, may have its own cost set.

2. **Hours.** The hours are the sum of the time between the start and end time of the routes, minus the wait time before the end of the routes. These hours are minimized in the optimization process. The *start time* of a route is defined as the moment of coupling a trailer to a truck and the *end time* of a route is the moment of decoupling. It may happen that a wait time is included before the trailer is decoupled from the truck, for example when the depot is closed. However, in practice, the decoupling

will still take place before the opening of the depot. Therefore, possible wait times before decoupling are excluded from the objective 'hours'. Some of the actions that are included are

- driving time,
- breaks, according to the drivers legislation,
- load and unload time of the vehicle,
- wait time,
- handling time.

Note that the hours are included in the plan costs as well. However, in plan costs they are included with weights according to the costs that are configured.

3. **Number of used trips.** The total number of routes that are used is minimized.
4. **Driving time.** The actual time the vehicle is moving, in seconds, is minimized. Note that the driving time is related to the distance and is therefore implicitly included in the plan costs.

These multiple objectives have, as already said before, an order of importance. For example, when two solutions are compared, they are first compared based on the number of planned transports in the solution. When this number is not equal, the solution with the most planned transports is chosen as starting point for the next iteration. In case the solutions have the same amount of planned transports, the second objective is considered, in this case the total plan costs. Again, the solutions are compared. In case the total plan costs are not the same, the solution with the least plan costs is chosen as starting point for the next iterations. In case they are not equal, the third objective is considered. This process continues as long as the solutions remain equal for the objectives. In case all objectives are the same, the best solution found in the previous iterations will be the starting point for the next iteration.

Note that the order of objectives has a great influence on the performance of the optimization process. The order can be determined in several ways. The customer, for whom the planning is made, may have a strong preference. For example, in case the costs per vehicle are very high, they need to minimize the number of used trips. Therefore, this objective should be high in the ordering of objectives. On the other hand, when testing the configurations, one may notice that there are many long wait times at customer locations in the planning. In this case, it might be helpful to move the objective 'total hours' up in the ordering.

For the experiments in this thesis, the order of the objectives remains fixed, as they are stated above. They are based on the objectives of one of two customers that are used during our tests. In future research, one might vary the order of the objectives as well.

Chapter 4

Minima-escaping Heuristics

The VRP can be very challenging to solve, especially when many attributes are included, such as multiple depots or heterogeneous fleet. Usually, the algorithms that are used consist of two phases:

- a construction phase, to find an initial feasible solution,
- and an improvement phase, to improve the solution quality of the initial feasible solution with respect to the objective in the model.

This chapter will mainly focus on the second part, the improvement phase. Most of the algorithms that are described require an initial feasible solution as input.

As already explained in Chapter 1, heuristics are needed for problems that are hard to solve. In case a heuristic is used, it is not guaranteed that the optimal solution will be found. To make sure that the heuristic that is used provides a near-optimal solution, the parameters of the heuristic need to be tuned for the instance that is solved. Because it can take a lot of time to search through the solution space, *metaheuristics* are introduced, that combine heuristics to efficiently search through the solution space. With metaheuristics, only a few solutions are explored, from which structural information is gathered, to efficiently search for near-optimal solutions. Therefore, metaheuristics aim to have a low computation time, but find near-optimal solutions. They are known for their guidance in the search process and are most of the time not problem- and case-specific (Blum & Roli, 2003). Because metaheuristic are most of the time not problem-specific, less parameter and variable tuning has to take place, compared to simple heuristics. Furthermore, in the metaheuristics there has to be a balance between diversification and intensification. *Diversification* implies exploring parts of the solution space that are unknown or not yet visited. On the other hand, *intensification* implies searching for solutions near good solutions that are already found in the previous steps of the search. Algorithms that combine the two make sure that many parts of the solution space are explored in detail (Glover & Laguna, 1997). For more information on intensification and diversification, we refer to Blum and Roli (2003), because they give a clear overview which summarizes multiple articles that include intensification and diversification.

Before the metaheuristics that are relevant for this research are described, we will discuss three local search methods in Section 4.1 that operate on small neighborhoods. The metaheuristics that operate on larger neighborhoods, and that are relevant for this thesis, will be discussed in Section 4.2. This section helps to understand the relation of the idea behind heuristics that make use of ruining and recreating a solution with the idea behind other metaheuristics, and is based on Chapter 4 of Toth and Vigo (2014). Next, in Section 4.3, a hybridization method is explored that combines several (meta)heuristics to create an algorithm that benefits from all the advantages of these (meta)heuristics. So far, the sections describe the metaheuristics purely theoretical and in general. Section 4.4 provides problem specific details about the most relevant metaheuristics that are used in this thesis. This section is used to provide insight into how to use and configure these metaheuristics

for the VRP. Section 4.4 concludes with the heuristic that is used for the experiments that are reported on in this thesis.

4.1 Local Search

There are several local search methods known that are used to improve a solution for a VRP. We will refer to these methods as local search steps or simple local search, as the methods that we use only involve small neighborhoods. We will describe three of these local search methods, to be able to use them in our research.

2-Opt

The first method we will describe is an edge exchange method, called *2-Opt*. With this method, two edges are deleted from the solution, which results in a broken route or tour. There is a unique way of reinserting the edges in a new place, such that a whole route or tour is constructed again (Cook et al., 1998). Usually, improvements are found in case a route crosses its own path.

CROSS-exchange

Another edge exchange method is *CROSS-exchange*. With this improvement method, two times two crossing edges are exchanged. Usually, improvements are found when two routes cross paths twice. This can be seen as executing 2-Opt twice, between two routes that cross paths.

Move

The last local search method that is defined is *move*, or sometimes called *relocate*. This method tries to find improvements by moving a group of tasks to another location in the solution. Note that the tasks that can be moved in our VRPO are delivery tasks, as all pickup tasks are located at the same depot. Hence, moving a delivery task to a different location implies moving a transport.

4.2 Metaheuristics: Local Search Class

The metaheuristics that are studied in this thesis are from the class of local search methods. The heuristics that belong to this class walk through the solution space by moving from a solution to a solution in its neighborhood (Toth & Vigo, 2014). They vary by, for example, having different definitions for the neighborhoods or when to accept new solutions. The reasoning for having these metaheuristics of the local search class, is to escape local optima and to avoid finding the same solutions over and over again (Toth & Vigo, 2014). Another class of metaheuristics is the class of population-based algorithms. Because the metaheuristics that are described in this thesis all belong to the local search class, we refer to the book by Toth and Vigo (2014) for more information about the population-based algorithms. Next, we give an overview of the most important meatheuristics, for this thesis, that belong to the local search class. Note that we assume that the problem considered is a minimization problem.

4.2.1 Simulated Annealing

Simulated annealing was first used by Kirkpatrick, Gelatt, and Vecchi (1983). Normally, local search methods only allow moves towards the optimal solution. Hence, only solutions with an improved objective value are accepted during the search. However, with simulated annealing, also solutions are accepted that move away from the optimal solution. The idea behind this method, is to get out of a local optimum and to search for the global optimum.

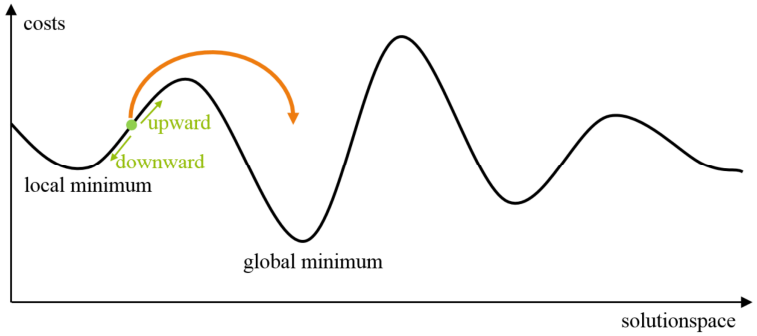


Figure 4.1: A figure to illustrate the principle of escaping a local minimum in the search for a global minimum. Normally, with local search only downward moves are accepted, in case a minimization problem is considered. However, with Simulated Annealing, upward moves are also accepted.

During the search, several solutions are found. We define a solution s_i as the solution found in the i^{th} iteration. Consider the solution s_{i+1} that is found in the next iteration. In case $f(s_{i+1}) \leq f(s_i)$, where f is used as a notation for the objective function, s_{i+1} will be the starting point for the next iteration. Note that the problem considered is a minimization problem. The other way around, in case $f(s_{i+1}) > f(s_i)$, s_{i+1} will be accepted as starting point for the next iteration with probability

$$\exp\left(-\frac{f(s_{i+1}) - f(s_i)}{t}\right). \tag{4.1}$$

Note that $f(s_{i+1}) - f(s_i)$ is always bigger than zero in this case and thus the probability of accepting a solution is always smaller than one. In (4.1), t is a parameter, often called 'temperature', that decreases over time with a cooling parameter c. Most of the time $t_{i+1} := t_i \cdot c$ for the i^{th}. Usually, this c is a fixed number within $(0, 1)$ and t_0 denotes the initial temperature. Both parameters, c and t_0, have to be tuned for the instance that is looked into. After a predefined number of iterations, the temperature will decrease, and the process continues with the next set of iterations. Therefore, simulated annealing is used as a framework with two levels: the outer and inner one. In the outer level, the temperature is varied with the cooling parameter c, and in the inner level a few iterations with this temperature are executed (Castro Martings & Sales Guerra Tsuzuki, 2014). Considering (4.1), one could notice that a small difference between $f(s_{i+1})$ and $f(s_i)$ implies a high probability of accepting s_{i+1}, in case t is fixed. Therefore, only small steps away from the local minimum are accepted. For $f(s_{i+1}) - f(s_i)$ fixed, a high temperature t implies a high probability of accepting s_{i+1} as a starting point for the next iteration. Therefore, at the beginning of the search it is more likely to accept solutions that move away from the local minimum. Simulated annealing can be seen as a metaheuristic that combines local search with diversification, where the diversification is introduced with a probability for accepting solutions that move away from local minima. For more information on simulated annealing and applications of this method, we refer to Castro Martings and Sales Guerra Tsuzuki (2014).

4.2.2 Tabu Search

Another metaheuristic that is in the class of local search methods, is *tabu search*. Within this metaheuristic, local search is executed and solutions found during the search are stored to avoid finding the same solutions over and over again. For a solution s_i, the search for a next solution s_{i+1} will be in the neighborhood of s_i. However, not all solutions in the neighborhood are considered as candidates for s_{i+1}, some of the solutions are on a *tabu*

list. Candidate-solutions s_{i+1} that have certain attributes are on this tabu list. This list may be altered during the search, by taking an attribute that is forbidden off this list after a certain criterium is met for this attribute. For example, in case a solution is accepted as best solution so far with an attribute that is similar or related to a tabu attribute. An example of such a tabu attribute can be high costs on arcs in the network. These arcs are not likely to be part of the optimal solution. Therefore, all candidate-solutions s_{i+1} that are in the neighborhood of s_i that contain such an arc with high costs, are not considered (Toth & Vigo, 2003). Tabu search may be seen as a framework for influencing the direction of the search for other solutions, just as simulated annealing.

4.2.3 Iterated Local Search

Whereas simulated annealing can be used as a local search framework for accepting solutions that are found with any local search procedure, *iterated local search* is a local search framework for accepting solutions that are found with any local search procedure after perturbing the previously found solution (Toth & Vigo, 2014). Starting with an initial solution, local search is applied till a stopping criterion is met, after which this solution is perturbed. With this stopping criterion comes an acceptance criterion of the new found solution. With this perturbation, a new starting point for local search is generated, provided that the perturbation is strong enough. Local search is used to intensify the search and the perturbation steps are executed to diversify the search.

The perturbation steps are very important: a strong perturbation will indicate an almost random restart for the algorithm, whereas a weak perturbation will be undone by the local search that follows. In the latter case, the perturbation step loses its effectiveness. The choice between strong perturbation and soft perturbation can be seen as strong and soft diversification in the search for the optimal solution.

4.2.4 Variable Neighborhood Search

A metaheuristic that is similar to iterated local search, is *Variable Neighborhood Search* (VNS). To escape local minima, it follows the same idea of perturbing the neighborhood. However, this perturbation step may be different for each iteration. For VNS the neighborhoods that are considered are of increasing complexity and size (Toth & Vigo, 2014) and form a family of neighborhoods (Pisinger & Ropke, 2007). The process starts with a feasible initial solution. A solution in the small neighborhood of this initial solution is chosen at random. Local search is applied till some stopping criterion is met and it is assumed that a local minimum is found. In case this local minimum is better than the best solution found so far, this local minimum is used as a starting point for the next iterations with the same neighborhood. In case the local minimum is not better than the best solution found so far, the process goes back to the previously found solution. However, a larger neighborhood is chosen until a better local minimum is found (Hansen & Mladenovic, 2001). For VNS intensification and diversification is used in two ways: applying local search followed by moving to a different neighborhood and by selecting different kinds of neighborhoods during the search in a predefined order.

4.2.5 Large Neighborhood Search

Another metaheuristic that perturbes the solutions that are found during the search is proposed by Shaw in 1997, called *Large Neighborhood Search* (LNS) (Shaw, 1997). This method is based on relaxation and re-optimization of solutions, which implicitly defines the neighborhood of this search. The search starts with a feasible initial solution. A part of this solution is relaxed, hence a part is deleted, after which a feasible solution is constructed again, to re-optimize the solution. This process continues, without using local search to intensify the search, which is the main difference between the previous metaheuristics and LNS. However, Shaw mentions in this paper that any search procedure can be used as reinsertion method, including local search.

In some way, LNS is related to VNS. In case reinserting the relaxed part of the solution does not provide a new feasible solution within a predefined computing time, LNS continues with a smaller neighborhood for relaxing during the next iteration. In case new solutions are found for a predefined number of iterations in a row, and the quality of these solutions is better than the best solution found so far, a larger neighborhood is used for the next iterations. With this procedure, intensification and diversification is included. Note that the method of relaxing the solution remains the same during the search, but the size of the neighborhood is decreased and increased. Therefore, different neighborhoods are used during the search, different regarding the size of the neighborhoods.

4.2.6 Ruin and Recreate

A local search method that is very similar to LNS proposed by Shaw (1997), is called *Ruin and Recreate* (R&R), proposed by Schrimpf et al. (2000). The idea of this method is the same as for LNS: a part of an initial or current solution is ruined, or relaxed as it is called in LNS, and the partial solution is recreated, or re-optimized. The goal of Schrimpf et al. (2000) is to use large neighborhoods to walk easily through the landscape of solutions in search of a global minimum.

There are two main differences between R&R and LNS:

- the methods to ruin, or relax, and recreate, or re-optimize, the solution in each iteration,
- when to accept a new found solution as starting point for the next iterations.

The first item implies that the neighborhoods that are searched through are different for both methods. The second item provides information about the framework that is used. R&R is also tested by Schrimpf et al. (2000) when all new found solutions are accepted during the search. This can be seen as a framework that is similar to simulated annealing, as solutions that have an objective function value that is worse than the current solution are also accepted.

4.3 Hybridization Methods

Even more advanced methods are known that find near-optimal solutions for complex problems: *hybrid methods*. A hybrid method combines several (meta)heuristics to create an algorithm that benefits from all the advantages of these (meta)heuristics. There are several ways to combine (meta)heuristics, for example by replacing a part of one (meta)heuristic by another (meta)heuristic. Another way to hybridize (meta)heuristics, is to use the output of one (meta)heuristic as the input for another (meta)heuristic. For more details about different hybrid methods, we refer to Toth and Vigo (2014) and Talbi (2013). The hybrid methods that are used in this thesis work with large neighborhoods. They are hybridizations of (meta)heuristics, mainly because structurally different neighborhoods are used during the search for near-optimal solution. This concept of hybridization relies on the idea behind the use of VNS, where the search alternates between neighborhoods to improve solutions. The difference between the hybrid method and VNS, is that the neighborhoods of VNS often increase (or decrease) in complexity, whereas the neighborhoods in the hybrid method are structurally different. This is explained in more detail in the next section. Each neighborhood would have a different distribution of final solutions, and therefore we talk about a hybridization. There is one hybrid metaheuristic that is frequently used and referred to in this thesis, called Adaptive Large Neighborhood Search. This method will be adapted and used for testing on the real-life data from ORTEC.

4.3.1 Adaptive Large Neighborhood Search

After both the papers of Shaw (1997) and Schrimpf et al. (2000) appeared in 1997 and 2000 respectively, Ropke and Pisinger made adaptions to the method proposed by Shaw. In

their method, called *Adaptive Large Neighborhood Search* (ALNS), they use several removal and insertion heuristics during the same search (Ropke & Pisinger, 2006a). This is the main difference compared to LNS and R&R, where only one single removal and insertion method are chosen during a search. Therefore, ALNS can be seen as a hybrid method. Performance differences between LNS and ALNS are only noticeable when large instances are investigated, and in that case ALNS performs better than LNS (Ropke & Pisinger, 2006a). The strength of ALNS is that it is able to adapt to different problems and instances, because of a possible learning layer. This layer is used to choose a removal and insertion method, based on their effectiveness during previous iterations. Heuristics that have a good performance will get a higher probability to get chosen in next iterations. In case a new solution is accepted or after a certain amount of iterations, the performance of the removal method is measured and updated. This method is known as *roulette wheel selection*. With the use of the roulette wheel selection, the neighborhoods that are explored are structurally different, as different removal and recreate methods are used. Whereas for VNS the neighborhoods are usually embedded, with ALNS they are structurally different. This is illustrated in Figure 4.2 (Pisinger & Ropke, 2007).

Because the performance of algorithms can be very problem specific, this learning layer will make the algorithm more robust. This is due to the fact that several removal and insertion heuristics can be used during one search, and the algorithm is adapting to the problem that is looked into. More information and details on the removal and insertion methods, is provided in Chapter 6. Ropke and Pisinger also found that a combination of heuristics with a good performance and a less good performance works better than only using heuristics with a good performance. This is probably due to the possibility to get out of a local minimum when accepting a solution that is worse than the currently found solution. When removing a part of this worse solution and rebuilding it, there is a possibility to come close to another, maybe better, local minimum.

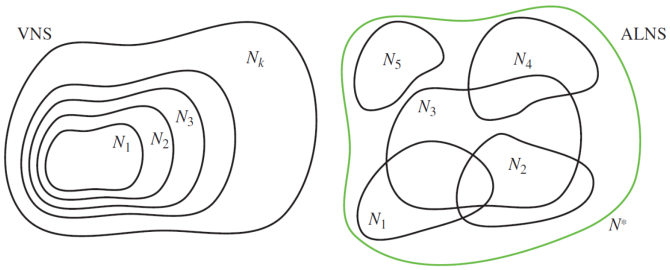


Figure 4.2: On the left the structure of neighborhoods is given for VNS, on the right for ALNS. For VNS, the neighborhoods are usually embedded and one structure is used. For ALNS the neighborhoods are structurally different defined by the corresponding search heuristics (Pisinger & Ropke, 2007).

4.4 Problem Specific: Local Search Metaheuristics for VRPs

Prior to this section, the local search metaheuristics are described in general, which allowed us to find the similarities and differences between the methods. However, they can be used for different optimization problems, for which the configurations may look different. Therefore, this section is included to explain how to utilize three of the local search metaheuristics that are most relevant for this thesis: LNS, R&R and ALNS. For each of these metaheuristics a pseudocode is given to illustrate the differences in more detail for the VRP. Recall that LNS, R&R and ALNS have in common that the solution is relaxed and re-optimized

during each iteration. In terms of a VRP, this indicates that transports that are planned in the solution are removed from the solution to be reinserted in a different place. In case a customer is removed from the solution, this indicates that all transports that need to be delivered at the customer location are removed from the solution.

4.4.1 Large Neighborhood Search

The first local search metaheuristic that we discuss in detail is LNS by Shaw (1997). An initial solution is constructed and it is relaxed by choosing a set of customers and remove them from the solution. This can be seen as a perturbation step to diversify the search. The removed customers are reinserted in the solution such that the costs are less than the costs of the initial solution. Which customers have to be removed from the solution is decided with a so-called *relatedness function*. First a random customer is selected to be removed from the solution, next a predefined number of customers that are most related to the first customer are removed.

For more details is referred to Shaw (1997), and this relatedness function will be described in more detail in Section 6.2. The method that is used to reinsert the removed customers is branch and bound. We will not go into detail about this reinsertion method, as we will not include it in our experiments. For more information we refer to Shaw (1997). Shaw (1997) proposed the LNS as an alternative for the use of local search, to escape local minima. The LNS is able to operate on large neighborhoods and Shaw (1997) states that the technique is completely general, because the relatedness function can be adapted to specific vehicle routing problems.

The pseudocode for LNS is given in Algorithm 1. After initializing the variables, a while loop is started that runs for a predefined amount of time. A transport is chosen at random and stored in the set of transports that will be removed, in line 3 and 4 respectively. Another while loop is included that runs till the number of transports that need to be removed is reached, line 5. The transports are chosen in the following way: a transport is randomly chosen from the set of transports that will be removed, called v (line 6,) and all other transports (not included in the set of transports that will be removed) are ranked according to their relatedness to this transport v (line 7). To include some randomness, a transport is added to the set of removals that is in a certain place in the ranking, influenced by a randomness parameter p and a randomly chosen number r (line 8-10). In case $p = 1$ the relatedness is ignored and with high values for p the relatedness is incorporated in the algorithm. With these r and p the LNS contains some diversification. When the transports are removed and reinserted with branch-and-bound (line 12), the acceptance criteria influences whether the new found solution will be the starting point of the next iterations or not.

4.4.2 Ruin and Recreate

In 2000 an article by Schrimpf et al. (2000) appeared with a method that can handle complex VRPs: R&R. Recall that LNS is comparable with R&R and that the main differences are in the methods that are used for removing and reinserting transports. Whereas LNS uses a relatedness function for removing transports, Schrimpf et al. (2000) propose three ruin methods:

- selecting one customer at random and remove a predefined number of neighbor customers,
- remove a predefined number of randomly selected customers,
- remove a predefined number of customers from a route that is selected at random.

Because the authors want to present the idea behind their R&R method, they use a simple insertion method called cheapest insertion, which is explained in more detail in Section 6.1. Another difference between LNS and R&R is when to accept a new found solution. Shaw (1997) accepts new solutions only when they have lower costs than the previously found

Algorithm 1: Large Neighborhood Search (VRP specific)

Input: An initial solution s_0. T are all transports in the network, T_r is the set of transports that are removed, p controls the randomness in the algorithm.

Output: The solution s^* with the best value for the objective function.

```
begin
    s* ⟵ s_0
    z ⟵ s_0
    T_r ⟵ ∅
1   while Computing time does not reach its maximum do
2       s := z
3       randomly choose v_0 ∈ T
4       T_r := {v_0}
5       while |T_r| < some number do
6           randomly choose v ∈ T_r
7           rank all T \ T_r with respect to their relatedness to v
8           randomly choose r ∈ [0, 1)
9           select transport w that is r^p of the way through the ranking
10          T_r := T_r ∪ w
        end while
11      remove all transports in T_r from s
12      reinsert all unplanned transports in s with branch-and-bound
13      if f(s) < f(s*) then
14          s* := s
15          z := s
        end if
    end while
end
```

solution, whereas Schrimpf et al. (2000) also test their R&R method when all new found solutions are accepted.

To see the difference with LNS, the pseudocode is included in Algorithm 2. As already explained, the removal method is different than with LNS and the acceptance criterion as well. Also, the stopping criterion is for R&R set to a maximum number of iterations, whereas in LNS the algorithms stops after a predefined running time. In this Algorithm 2, the first ruin method that is described above is included. For the other two methods, the difference in pseudocode will be in line 5-7. With R&R a neighborhood N_v for transport v need to specified and will be explored.

4.4.3 Adaptive Large Neighborhood Search

The previous two methaheuristics only include one removal and one reinsertion method for the transports. However, in ALNS it is possible to choose a different removal and reinsertion method in each iteration of the search with a roulette wheel. The different methods will be described in detail in Chapter 6.

A pseudocode for ALNS is given in Algorithm 3. The main difference with LNS and R&R is that the roulette wheel is included to keep scores for the removal and insertion methods. First, a removal and insertion method are chosen (line 3) and the transports are removed according to the removal method (line 4-7). Afterwards, the removed transports are reinserted again according to the chosen method (line 8). The new found solution is accepted or not, according to the acceptance criterion, and the scores for the roulette wheel are updated. Note that the updating of the scores w in line 12 does not only include updating the scores according to the performance of the removal and insertion methods. After 100 iterations of removing and recreating the solutions, the scores are set to zero.

Algorithm 2: Ruin and Recreate (VRP specific)

Input: An initial solution s_0. T are all transports in the network, T_r is the set of transports that are removed, N_v is the neighborhood of $v \in T$.

Output: The solution s^* with the best value for the objective function.

```
begin
    s* ⟵ s_0
    z ⟵ s_0
    T_r ⟵ ∅
1   while The maximum number of iterations is not reached do
2       s := z
3       randomly choose v_0 ∈ T
4       T_r := {v_0}
5       while |T_r| < some number do
6           randomly choose v ∈ N_{v_0}
7           T_r := T_r ∪ v
        end while
8       remove all transports in T_r from s
9       reinsert all unplanned transports in s with cheapest insertion
10      if The acceptance criterion is met then
11          s* := s
12          z := s
        end if
    end while
end
```

This is needed to keep diversifying and intensifying the search. In case the scores are not set to zero after a predefined number of iterations, the methods that perform very well are chosen with an increasing probability. Further in the search, this means that the same methods will be chosen over and over again. This may cause the hybrid method to get stuck at a local minimum. This can be avoided by setting the scores to zero after a specific number of iterations.

4.4.4 ORTEC Adaptive Large Neighborhood Search

The hybrid method that is used in our experiments is inspired by LNS, R&R and ALNS. It can be seen as ALNS with some adjustments. Some parts will be left out of the hybrid method and some new components will be added.

First of all, we will look at differences in the configuration between the ALNS of Ropke and Pisinger and the one that is configured at ORTEC, before a pseudocode is given. To clarify these differences, we will refer to this pseudocode during the explanation of the differences.

- The first difference is about the objective function. In Ropke and Pisinger (2006b) the objective function contains equal weights for distance and work time of the vehicles. As already explained before, the objective function for ORTEC has an ordering in the objectives.
- Another difference that we will discuss concerns the scores of the removal and recreate methods. With the original ALNS the scores are updated separately for both the removal and recreate methods. Within the software of ORTEC the scores are updated according to a so-called 'method', which includes both a removal and a recreate method. The performance is measured according to the quality of the new found solution that is constructed with a recreate method after a removal method is used. Either the solution has a better objective value than the currently best found solu-

Algorithm 3: Adaptive Large Neighborhood Search (VRP specific)

Input: An initial solution s_0. T are all transports in the network, T_r is the set of transports that are removed, w contains the weights for both choosing removal and insertion methods, t_n is the number of transports that is removed, $[a, b]$ is an interval of possible numbers of transports that are removed.

Output: The solution s^* with the best value for the objective function.

```
   begin
       s* ⟵ s_0
       z ⟵ s_0
       T_r ⟵ ∅
1      while The maximum number of iterations is not reached do
2          s := z
3          choose a random number nt from [a, b]
4          choose a removal and insertion method according to scores w
5          while |T_r| < t_n do
6              remove transports according to the removal method
7              update T_r
           end while
8          remove all transports in T_r from s
9          reinsert all unplanned transports in s with the chosen insertion method
10         if The acceptance criterion is met then
11             s* := s
12             z := s
           end if
13         update the scores w
       end while
   end
```

tion, which influences the score in a positive way, or the objective value is worse, which decreases the score. Therefore, in case three removal methods and three recreate methods are used, there are nine methods that obtain a score. Hence, instead of two roulette wheels for both the removal and recreate method, we incorporate one roulette wheel with all combinations of removal and recreate methods. Note that the difference in Algorithm 4 are only in the definition of the scores w, which are updated in line 19.

- A consequence of the previous point will be that our roulette wheel needs more iterations to 'learn', because more methods are included. Hence, the maximum number of iterations in line 3 of Algorithm 4 will be higher.
- Most of the removal and recreate methods are configured in a different way. For example, Ropke and Pisinger (2006b) include more randomness in the removal methods than is used in the software of ORTEC. Furthermore, insertion is done based on the objective function for Ropke and Pisinger, whereas within the software of ORTEC it is possible to specify the objective for each insertion method separately. More details about the differences in removal and recreate methods will be described in Chapter 6.
- The number of transports that are removed and reinserted during the search are different. Pisinger and Ropke (2007) use an interval from which in each iteration a random number is chosen. This number will be the number of transports that are removed. Within the software of ORTEC, one needs to specify upfront the number of transports that are removed. This is done for each removal method separately. Therefore, no randomness is included in the number of transports that are removed.

Algorithm 4: ORTEC Adaptive Large Neighborhood Search

Input: T are all transports in the network, T_r is the set of transports that are removed, w contains the combined weights for removal and insertion methods, t_n is the number of transports that is removed, P contains a few numbers of transports that can be removed, t is a threshold.

Output: The solution s^* with the best value for the objective function.

```
begin
    T_r ⟵ ∅
1   Construct an initial solution s_0
    s* ⟵ s_0
    z ⟵ s_0
2   for All numbers t_n ∈ P do
3       while The maximum number of iterations is not reached do
4           s := z
5           choose a method, including a removal and a insertion method, according
             to the combined scores w
6           while |T_r| < t_n do
7               remove transports according to the removal method
8               update T_r
            end while
9           remove all transports in T_r from s
10          reinsert all unplanned transports in s with the chosen insertion method
11          if f(s) < f(s*) then
12              s* := s
13              z := s
            end if
14          else if f(s) < t · f(s*) then
15              run improvement iterations with local search
16              if f(s) < f(s*) then
17                  s* := s
18                  z := s
                end if
            end if
19          update the scores w
        end while
    end for
end
```

In Algorithm 4 this is incorporated with a for loop that starts in line 2. For example, each of the numbers in $P = \{30, 45, 60\}$ represents a number of transports that needs to be removed. The algorithm starts with 30 transports that are removed, and a roulette wheel with a specific number of iterations is executed. When this maximum number of iterations is reached, the algorithm continues with the next number of transports that needs to be removed: 45 transports.

- Also the acceptance criteria is different. A simulated annealing acceptance framework is used by (Ropke & Pisinger, 2006b) where at the beginning of the search, besides solutions with a better objective value, also solutions with a worse objective value are accepted as a starting point for the next iterations. Within the software of ORTEC we can only temporarily accept solutions with a worse objective value, with the use of a threshold, which can be seen in Algorithm 4 line 11-18. For example, when this threshold t is set to 1.05 and the solution is at most 5% worse than the best solution found so far after reinserting the removed transports, an improvement phase starts

which takes this solution as starting point, see line 14-18. In case the solution is still worse than the best found solution after the improvement phase, the solution is not accepted. However, in case improvements are found and the solution has a better objective value than the best solution found so far, the solution is accepted and used as starting point for next iterations, line 16-18.

- As explained in the previous point, an improvement phase can be included with the help of a threshold in the software of ORTEC. For example, local search heuristic can be executed for a few iterations to explore the neighborhood of worse solutions than the best solution found so far. This helps to diversify and intensify the search. Local search heuristics are an addition to the original ALNS from Ropke and Pisinger.
- The last difference between the original ALNS and the one that is implemented at ORTEC, is that we need to specify the construction for the initial solution as well, see line 1 in Algorithm 4. The initial solution is not an input for our method, as is the case for the original ALNS. Ropke and Pisinger (2006b) do note that they use a regret insertion method for the construction of the initial solution, more about this is included in Chapter 6. However, they do not incorporate the influence of the initial solution on the performance of their ALNS. We will test this influence and therefore the initial solution is not used as an input for our heuristic, but it is a component of the heuristic.

To summarize, there are difference regarding the configuration of the objective function, removal and recreate methods, the roulette wheel an the acceptance criteria. A consequence is that we are not able to remove transports in the same way Rope and Pisinger do in their ALNS. However, additions to the ALNS are the incorporation of the construction method for the initial solution and local search methods to intensify the search after removing and reinserting transports.

Chapter 5

Introduction ORTEC and their Software

The research for this thesis is carried out at the company ORTEC, short for Operations Research Technology. They provide optimization software and analytical solutions to all sorts of companies. One of the software programs that ORTEC provides is called *ORTEC Routing and Dispatch* (ORD), which is used for this research. Several customers use this software, including retailers, shippers and logistic service providers. Two unique features of ORD are real-time event management with alerts for violated planning restrictions and the possibility for multiple planners to work at the same planning, at the same time. A more detailed description of ORD and its features is given in the next paragraph, followed by an overview of the structure for the configuration of the algorithms that are used.

5.1 Automated Planning Process

ORD has the functionality of a two step optimization process: route planning and resource planning. For the automated process for route planning, a planner usually wants to select a couple of routes and transports. In this way, the solver for this process finds the planning that provides the lowest KPIs for the selected routes and transport. To plan automatically, transports and routes are selected in their grids and the function for the automated process is started by a click on the automated planning button in ORD. The solver has been created by ORTEC and is called CVRS (COMTEC Vehicle Routing Service). The second optimization step would be to assign resources, hence drivers, to these constructed routes. Another solver is implemented at ORTEC to do so. However, the focus of this thesis is on route planning only and therefore we will not provide more information about the resource planning process.

5.1.1 Input for ORD

As already mentioned in the introduction of this chapter, it is possible in ORD to plan transports manually on routes. This is done with a drag-and-drop functionality, where transports are selected and dragged to the right route. The planner chooses in what order each task is executed, for example in what way the transports are loaded in the vehicle. In this thesis we will only focus on the automated planning process and we will not go into detail about the manual planning process. However, to provide extra information on the features and to get an idea of how planners use ORD, Appendix A is included.

To be able to plan transports on routes, manually or with the automated process, a database including all information about the data set is needed. This includes addresses of the depots and customers, the drivers and vehicles that can be used in the planning, restrictions on the vehicles and drivers, opening hours and restrictions of the depots and

customer locations and a lot more. The automated planning process will take into account the restrictions when searching for the best planning.

In what way the automated planning process is executed needs to be configured by consultants of ORTEC. Along with settings that contain information about the restrictions that are taken into account, such as congestion, a configuration of the algorithm that is executed needs to be provided as input as well. This configuration of the algorithm is known at ORTEC under many names, including a 'script' and 'template'. In this thesis, we will refer to it as a 'configuration', which implies that it contains information on the algorithm that is executed. However, this should not be confused with the actual configuration of all the heuristics and functions that are implemented at ORTEC. To clarify this, the next section is included.

5.1.2 Congifurations for ORD

What heuristics need to be executed and in what way, configurations are used as an input for ORD. In these configurations the objectives of the model are defined, followed by all the different algorithms that need to be executed and the corresponding settings of these algorithms. The configurations are XML files that follow a specific structure using different templates that consist of several algorithms. In the configuration, there are no algorithms described, they are only executed. One can see this as functions that are executed, with parameters and variables as input.

As already mentioned, the configuration that runs to construct the planning automatically follows a specific structure. This structure is not included for technical reasons, the XML file can be read perfectly by the solver when there is no structure at all. The structure is rather used for the planners to be able to read and compare different configurations. To illustrate the idea, an example of a configuration is provided in Listing 5.1. Note that the configuration starts with defining the objectives, in which the order is important as is described in Chapter 3. After the objectives, several templates are defined, which are executed in a specific order. An overview with more details about the structure is given next, as we will walk through the lines of the example configuration in Listing 5.1.

1. *Define the Objectives.*
   The configuration starts with defining the global settings at line 4, including all objectives that are considered for the optimization process. Several objectives could be defined that follow an order of importance, as described in Chapter 3. Hence, when there are more solutions with the same value for the first objective, the second objective is considered to determine the best solution etc. In Lising 5.1, the number of planned tasks is maximized, and the distance, plan costs and hours are minimized.

2. *Determine the Planning Process.*
   Right after the global settings an `InsertTasksIntoTrips` strategy is used where the actual planning procedure is described. This template will be executed by the process that runs for the automatic planning. All templates that are executed have to be written down in this main template.

   Most common is to start with the construction of an initial solution, line 14, and improve it with local search, line 15. It may happen that not all the transports are planned during the construction phase, and that the local search phase will create extra space for new transports to be inserted, line 16. In that case, a second construction part is needed after the local search. All these algorithms are described in templates with the same strategy name. These templates are defined right below the main template, from line 19 till line 42.

3. *Templates.*
   Templates make sure that the algorithms are used in a proper way during the planning process. There is for example a template for the construction, line 19-22. This construction part contains all insertions of transports in routes, as far as this is possible. The construction is built with several algorithms. For example, one could think

of sorting transports and insert the ones that are hard to assign first. One could build this template for the construction with other sub-templates, one for each step in the construction. In this way, when proper names are given to the sub-templates, the steps for constructing the initial solution will be clear in just one quick look at the script.

To clarify the structure and the usage of sub-templates, the example configuration in Listing 5.1 is looked into in more detail. As said before, the planning process is defined in lines 13-17. The template for the construction phase is executed in line 14, and the template is given in lines 19-22. This construction starts with the planning of seed tasks with sub-template `PlanSeedTasks`, lines 24-33. In here the transports, or tasks, are sorted in groups. Because we want to insert one seed per route, each group will only include one task. Note that one transport consists of two tasks: a pickup and a delivery task. The first group contains the task that is furthest away from the depot, and the last group contains the task that is closest to the depot. In this way, all routes get assigned one transport as a seed transport, with transports that are far away from the depot. They are inserted into the routes with parallel cheapest insertion, cheapest when calculating the extra distance that needs to be driven. This attribute `EstimateWith` in line 30 is used to specify the *estimator* that is used to estimate the extra costs for inserting a transport. Because only empty trips are considered, `OnlyEmptyTrips` is set `true`, all routes get assigned one transport.

After the planning of the seed transports, the construction phase continues with the planning of the remaining transports, line 21, in the `PlanRemainingTasks` sub-template. This sub-template is defined after the `PlanSeedTasks` sub-template, lines 35-37. Here the routes are filled with transports, using parallel cheapest insertion, while taking distance into consideration for inserting.

Now that the construction phase is finished, the planning process continues with a local search phase in line 15. The template for this phase is given in lines 39-42. For the local search 2-Opt and CROSS exchange are used, as described in Section 4.1. This template is finished, hence the planning process continues with the planning of transports that remained unplanned, line 16. For this, the same sub-template is used as for the construction phase, given in lines 35-37.

In case more sophisticated methods are used to construct and improve an initial solution, these configurations can become very long. It is necessary to cut the configuration in many templates and sub-templates, to give the configuration the structure that it needs to be read.

5.1.3 Output from ORD

After an automated planning process is started and a stopping criterion is met, ORD will provide output. Whereas before the planning the selected routes were empty, now they will be filled with transports that are planned on the routes. The KPIs, Key Performance Indicators, can be seen for the planning as well. Some of them are optional and differ per customer.

However, for this thesis we will not work with the outer framework that is used by planners. We will use the XML files that are produced by the software. These files contain information about the settings that were configured, the transports and routes (with their ID's) that are used as input, which transports are planned on which routes after the planning is made and the procedure of finding the final planning. However, as this information is stored in XML files, for plannings that have a long computing time the XML files are very big. Therefore, they can be hard to read. Nevertheless, we prefer these XML files over the outer framework that is used by the planners, because much more information is stored in these files. For example, when the transports are divided in groups and they are inserted group by group, the XML files store which transports are in the groups and in what order they are inserted. Furthermore, the objectives are given as KPIs to keep track of the solution quality.

Listing 5.1: An example of a configuration that is used in CVRS to plan transports on routes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<RootTag>

   <GlobalSettings>
      <Objective>
         <Element direction="maximize" name="#TA"/>
         <Element direction="minimize" name="d"/>
         <Element direction="minimize" name="$p"/>
         <Element direction="minimize" name="h"/>
      </Objective>
   </GlobalSettings>

   <Template Strategy="InsertTasksIntoTrips" Part="toInsertInBase"
   command="CAlgBuildCombinedStrategy">
      <CAlgBuildStrategyFromSetting Strategy="Construction"/>
      <CAlgBuildStrategyFromSetting Strategy="LocalSearch"/>
      <CAlgBuildStrategyFromSetting Strategy="PlanRemainingTasks"/>
   </Template>

   <Template Strategy="Construction" command="CAlgBuildStrategyFromSetting">
      <CAlgBuildStrategyFromSetting Strategy="PlanSeedTasks"/>
      <CAlgBuildStrategyFromSetting Strategy="PlanRemainingTasks"/>
   </Template>

   <Template Strategy="PlanSeedTasks" command="CAlgBuildStrategyFromSetting">
      <CAlgDivideTasksInBatches MaxBatchSize="1">
         <SortingCriteria>
            <Criterion direction="decreasing"
            name="DistanceClosestDepotToTask"/>
         </SortingCriteria>
         <Template Part="Construction">
            <CAlgParallelCheapestInsertion EstimateWith="Distance"
            OnlyEmptyTrips="true"/>
         </Template>
      </CAlgDivideTasksInBatches>
   </Template>

   <Template Strategy="PlanRemainingTasks"
   command="CAlgBuildStrategyFromSetting">
      <CAlgParallelCheapestInsertion EstimateWith="Distance"/>
   </Template>

   <Template Strategy="LocalSearch" command="CAlgBuildStrategyFromSetting">
      <CAlg2OptInCurrentSolution/>
      <CAlgCROSSExchangeInCurrentSolution/>
   </Template>

</RootTag>
```

Chapter 6

Components of the Hybrid Method

In Section 4.4.4 we have introduced the framework of the hybrid method that is tested on in this thesis report. It can be seen as a variant of the ALNS (Ropke & Pisinger, 2006a) where the component of the initial solution is tested in addition as well. Furthermore, local search is included to improve the solutions that are found during the ruining and recreate phases of the hybrid method. Recall Algorithm 4 and the different components that can be configured

- initial solution, Section 6.1,
- removal methods, Section 6.2,
- recreate methods, Section 6.3,
- acceptance criteria, Section 6.4.

The structure of this chapter is as follows: each component is discussed in a different section where first an overview of the literature is given, related to the component, followed by an overview of the performance as described in the literature. These performance overviews only include the performance within a framework such as our hybrid method. It is used as an overview of the usage of each component in the literature, rather than, for example, the performance of construction methods for the initial solution in general.

6.1 Initial Solution

Before we can remove and reinsert certain transports, an initial solution needs to be constructed. Because there are many insertion heuristics, there are many ways of finding an initial solution. For example, simple heuristics can be used, such as a random approach, or one can choose a more sophisticated heuristic, such as regret insertion. The quality of the initial solution depends heavily on the choice of the insertion heuristic. However, with the hybrid method that is used in this thesis, this initial solution will be partially ruined and recreate after it is created. Therefore, one can question the need of having an initial solution of good quality. In this section, an overview of insertion methods that are used in the literature, when using metaheuristics such as LNS and ALNS, is given in Section 6.1.1. The performance of these methods that is described in the literature, is given in Section 6.1.2.

6.1.1 Literature Overview

In most articles that review (a variant of) metaheuristics that use removal and recreate methods, the construction of the initial solution does not get much attention. They mainly focus on which ruin and recreate methods to choose. Some articles do not even mention how

to construct the initial solution, such as Ropke and Pisinger (2006b) and Ahuja, Ergun, Orlin, and Punnen (2002). They only provide the metaheuristic itself, where a feasible initial solution is required as input.

There are two categories for insertion methods: *sequential* and *parallel insertion methods*. The categories describe in what way the routes are built. Besides these categories, methods for the construction of the initial solution may be categorized in a different way: the ones that use a single insertion heuristic, or that use additional heuristics as well, such as local search.

- **Sequential insertion.** For sequential insertion, a route starts with a seed customer and other customers are added to the route until the maximum capacity is reached or the route has a scheduled time horizon. The seed customer can be chosen based on several objectives. Most common is to choose a customer that is farthest away from the depot, or that has the first deadline of service. For the other customers that need to be inserted, one may sort the customers on distance that needs to be driven from the seed customer. Solomon states that this method is stable and will perform well on practical problems, mainly with tight time windows (Solomon, 1987). This may be explained by the fact that one could insert seed customers based on the tightness of the time windows. Unfortunately, there is also a disadvantage when using sequential insertion. At the end of the insertion process, it is very likely that the unrouted customers are scattered all over the network. Therefore, the distances between these customers may be very large and the resulting routes may be very inefficient, both regarding the time schedule and distance. For more details and methods about sequential insertion, we refer to Solomon (1987).
- **Parallel insertion.** To overcome this problem, Potvin and Rousseau (1993) create a method where routes are not built in sequence, so one by one, but rather in parallel. Therefore, this method is called parallel insertion. A predefined number of routes get a seed customer. For all unrouted customers it is calculated what the cost will be for inserting them in each one of these seeded routes. The minimum cost for each customer is derived from this, and the customer which causes the least extra costs will be inserted first. Potvin and Rousseau (1993) state that for instances where the customers are clustered, sequential insertion performs best, and for instances with some randomness in the distribution of customers, parallel insertion outperforms sequential insertion. The reason behind this, is that for clustered instances sequential insertion chooses a seed customer within a cluster, joins all customers in this cluster, and starts a new route with a seed customer in another cluster.

Now that the two categories of insertion methods are explained, we can look into some of the insertion methods. There are simple insertion methods, such as cheapest insertion, but also insertion methods that include some way of sorting the transports before they are inserted, such as regret insertion.

Cheapest Insertion

The first insertion method that is looked into is called *cheapest insertion*. In the literature this insertion method is also known as best insertion (Schrimpf et al., 2000), (basic) greedy insertion (Pisinger & Ropke, 2007) or least-cost insertion (Azi, Gendreau, & Potvin, 2010).

Cheapest insertion may be used as a sequential insertion method as well as a parallel insertion method. Cheapest, in this case, can be measured in many ways. For example, one could take into account the additional distance that needs to be driven, the additional driving time, the additional working hours, or for example a combination of the previous measurements. Note that, in case all routes have the same initial properties, such as starting time or starting location of the route, all costs for inserting transports will be the same. Therefore, routes need to be initialized with some seed transport, or the first transport that is inserted will be chosen at random.

Regret Insertion

Another, more sophisticated method is called *regret insertion.* It uses a 'look-ahead' strategy to be able to see whether a transport needs to be inserted or not (Potvin & Rousseau, 1993). This is done by considering the extra costs when a transport is inserted in the second best place compared to its best place. When this cost is large and the transport is not inserted in the best place, because another transport is planned on this position, you regret that you did not insert it. Hence, when the 'regret' is large, the transport needs to be inserted first.

Let Δf_v^q be the change in the objective function f for inserting transport v at the best place in the q^{th} cheapest route. For example, a regret-2 implies the difference in the objective value for inserting a transport in its second best place instead of the best place. Hence, the regret-2 value for a transport v will be

$$\Delta f_v^2 - \Delta f_v^1.$$

In each iteration, only one route will be modified. Therefore, only the costs for transports for which the best or second-best place is in this modified route, need to be recalculated. This insertion method belongs to the parallel category of insertion methods, because all routes are considered for new insertion at each iteration.

6.1.2 Performance

In most metaheuristics that are studied in this thesis, a simple heuristic is used for the construction of the initial solution, such as random insertion or cheapest (parallel or sequential) insertion. Shaw (1997) chose to plan all customers on a different route, which results in 100 routes for 100 customers. He tested his metaheuristic on Solomon instances, but for more realistic VRPs it is common to have a limited number of vehicles. Therefore, it may not be possible to insert each customer in a different route.

Another simple construction method, is to randomly insert the transports in routes. Lin (1965) and Jaszkiewicz and Kominek (2003) use a randomly created initial solution. This method finds an initial solution very fast, because no calculations need to be made for the choice of inserting a certain transport. However, it might be hard to find a feasible initial solution when, for example, many transports need to be inserted on a limited number of routes.

Cheapest insertion is used by, for example, Azi et al. (2010), Häll and Peterson (2013) and Schrimpf et al. (2000). Most of the time, cheapest is measured with additional detour distance for inserting a transport. Schrimpf et al. (2000) state that cheapest insertion provides an initial solution of better quality, and therefore their metaheuristic converges faster than when random insertion is used.

Pisinger and Ropke (2007) use the more sophisticated insertion heuristic regret insertion. This method performs well on problems with hard to plan transports, because these transports are inserted at the beginning of the construction.

Besides these single insertion heuristics, there is also a method provided to minimize the number of vehicles that is used, before using a metaheuristic to improve the initial solution. This method is proposed by Ropke and Pisinger (2006a). For more details on this method, we refer to Appendix B.

Only a few articles mention how to cope with infeasible solutions. Schrimpf et al. (2000) use an additional vehicle in case a transport can not be inserted, due to capacity or time window constraints. In case the number of vehicles that may be used is limited, Azi et al. (2010) propose to insert the unplanned transports using a recreate method that is used during the recreate stage of the metaheuristics that are described in this thesis.

More about the performance of the construction method for the initial solution, and information on how to cope with infeasible solutions, can be found in Appendix B.

The results of the metaheuristics, that are explored in the literature, regarding the choice for the construction of the initial solution, are not mentioned. However, we think that the choice for the construction of the initial solution is of great importance regarding

the performance of the ruin and recreate iterations. Quick and simple insertion heuristics do not provide an initial solution of good quality, but is this needed in the metaheuristics that are considered in this thesis? The initial solution gets ruined soon after it has been created, so using a complex and time consuming insertion heuristic may be seen as a waste of time. However, when working with realistic VRP, it can be hard to find a feasible initial solution. When working with a simple heuristic, you may not be able to plan all transports. Therefore, it may happen that you are forced to use a time consuming insertion method, in case a feasible initial solution is needed.

6.2 Removal Methods

After an initial solution is constructed, the next step is to ruin this solution by applying a removal method. As described in Chapter 4, for the LNS heuristic only one removal method is chosen to be executed, whereas for the ALNS heuristic multiple removal methods can be chosen. Before the performance of the removal methods is discussed in Section 6.2.3, a description of the methods is given in Section 6.2.1.

6.2.1 Literature Overview

There are many ways to remove transports from a solution. This section gives an overview and description for eight removal methods.

Related Removal

When transports need to be removed from a solution, it would be wise to remove transports that are similar. Because, if transports are removed that are very different, the new solution may be of the same quality as the current solution, as it is hard to shuffle the transports around. However, transports that are somehow related to each other, may be easy to interchange, which may lead to a better solution. 'Similar transports' can refer to multiple measures, such as transports being geographically close to each other or having the same time windows. Shaw (1997) proposed to use a method to remove transports where the relation between transports in measured with a *relatedness function*. This provides a number between 0 and 1 that indicates the relatedness between two transports. The function consists of four variables that contribute to a relatedness, and four parameters that indicate the importance of each relatedness variable. The relatedness variables that are used by Shaw are based on

- the costs of getting from one customer to the other, weighted with α,
- the similarity of the time windows for both transports, weighted with β,
- whether the customers are served by the same vehicle or not, weighted with γ,
- whether two transports have similar load weights, weighted with δ.

The relatedness variables need to be normalized and the parameters α, β, γ and δ need to be chosen between 0 and 1. The closer these parameters are chosen to 1, the more they affect the relatedness function. Combining all relatedness variables with their weights in a function results in a relatedness function, $\mathcal{R}(i,j)$, between two transports i and j,

$$\frac{1}{\mathcal{R}(i,j)} = \alpha d_{ij} + \beta\left|t_i - t_j\right| + \gamma T_{ij} + \delta\left|q_i - q_j\right|. \tag{6.1}$$

Note that the relatedness function is a fraction, because all variables and parameters have values between 0 and 1. Hence, a lower value of $1/\mathcal{R}_{ij}$, indicated a higher relatedness. In case a VRPTW is considered, d_{ij} represents the travel distance from the delivery location of transport j to the delivery location of transport j, $\left|t_i - t_j\right|$ is the absolute difference in start time of service between delivering transport i and j, T_{ij} is a boolean that equals 1

in case transport i and j are scheduled on the same vehicle, and $\left|q_i - q_j\right|$ is the absolute difference of quantity of goods between transport i and j.

Shaw used $\alpha = 0.75$, $\beta = 0.1$, $\gamma = 1$, $\delta = 0.1$ in his test. Hence, the geographical closeness and whether the transports are scheduled on the same vehicle contribute most to the relatedness between the transports.

A pseudo-code for how to use the relatedness function when removing transports is included in Appendix B. Shaw (1997) is not the only one that used this related removal, also Ropke and Pisinger (2006a) and Azi et al. (2010) use the relatedness function with some adjustments. More information is included in Appendix B as well.

The configuration at ORTEC is done in a different way for related removal. To decrease the computing time for related removal, only the distance between transports is taken into account in the relatedness function.

Random Removal

Another removal method that is frequently used in the literature, is *random removal*, proposed by Schrimpf et al. (2000). For this removal method, a predefined number of transports is selected at random and they are removed from the solution. This method can be seen as a global method, because the whole solution can be affected. This is mainly due to the random choice for the removals (Schneider & Kirkpatrick, 2006). The random removal is used in Schrimpf et al. (2000), Ropke and Pisinger (2006a), Ropke and Pisinger (2006b), Pisinger and Ropke (2007) and Azi et al. (2010).

Worst Removal

When a *worst removal* method is used, transports are removed that come with, for example, very high costs. One could consider the difference between the solution cost with and without the transports that is considered to be removed, proposed by Ropke and Pisinger (2006a). Hence, define the cost of a transport i as

$$cost(i, s) = f(s) - f_{-i}(s),$$

where s is the solution that is considered and $f(s)$ represents the cost of the solution. This $f(s)$ can for example be chosen as the solution quality, the objective value, or simply the total distance that is driven. In the latter case, $cost(i, s)$ can be seen as the detour that is needed to visit the location for transport i. These costs are now ordered in a descending way, such that the worst removals can be chosen from the top of the ordering. To make sure that not the same transports are chosen over and over again, a randomization parameter is included as well. For more details is referred to Ropke and Pisinger (2006a). Using the worst removal method, transports that are very expensive in the current solution are placed at cheaper positions, as if they were planned in the wrong position. This method can be seen as a global method as well as a local method. It can be the case that in a certain area there are many transports with high costs. Therefore, the worst removal will remove transports that are in the same area, and thus worst removal operates on a local level. On the other hand, the worst placed transports may be scattered all over the network and removing them will result in more global changes.

Historical-based Removal

So far, the removals look at the solution at its current state, none of the methods take historical information into account. Therefore, Ropke and Pisinger (2006b) propose two versions of a *historical-based removal*: *neighbor graph removal* and *request graph removal*. Historical information is used, for example, by storing scores for each transport. An example is to store how many times a transport is in the same route as an other transport, in all best found solutions so far. When a new solution is found, a transport that has a low score may be placed in an unsuitable route. Improvements may be found when this transport is removed form the solution and reinserted in some other place. More information about

these removal methods is included in Appendix B. However, Ropke and Pisinger found that this method does not perform well, because of the lack of diversification. The changes are based on the best found solutions so far, which may result in small changes only. Therefore, they propose to use these methods for the relatedness of two request in related removal. This removal method is not available in the software of ORTEC.

Cluster Removal

The *cluster removal* method is proposed by Ropke and Pisinger (2006b). First a route is selected at random and the transports on this route are divided into two clusters, based on distance. One of the clusters is chosen at random and the transports in this cluster are removed from the solution. In case the number of removed transports is less than is desired, another route is chosen at random and the process starts again. This removal method can be seen as a subclass of the related removal that is stated above, because transports that are related, via a route and distance, are removed.

In the software of ORTEC, two versions of cluster removal can be used: *cluster random removal* and *cluster worst removal*. For cluster random removal, a transport is randomly chosen as the first one that will be removed. Next, all transports that are in its direct neighborhood will be removed as well. For cluster worst removal, the initial transport that is removed will be the most expensive on, instead of a randomly chosen transports. A predefined number of transports that need to be removed is given as input, along with the number of clusters. This defines how many transports are removed per cluster, which indicates the definition of the direct neighborhood. Hence, the clusters are not components of one route, as is the case for Ropke and Pisinger.

Sequential Removal

The removal method proposed by Schrimpf et al. (2000) is very similar to cluster removal and is called *sequential removal*. In this removal method, a route is chosen at random and a predefined number of transports is removed from this route. The transports have to be in sequence. Cluster removal can be seen as several sequential removals executed at the same iteration.

In the software that is used for this research, it is not possible to remove a sequence of transports from a route. However, another removal method is included that removes all the transports on a route, called *trip removal*. As input, this method requires the number of trips, or routes, that need to be removed. Almost empty trips are preferred above full trips when selecting candidates to be removed.

Time-oriented Removal

A version of the related removal is mentioned by Shaw (1997) and executed by Pisinger and Ropke (2007). Shaw uses the absolute difference in start time of service between to transports, the variable with parameter β, as an element of a relatedness function, whereas Pisinger and Ropke use this single element as a whole new removal method. They call it the *time-oriented removal* and takes into account the difference between the pickup time and delivery time of two transports. The time-oriented distance between two transports is defined as

$$\Delta t_{ij} = \left|t_{p_i} - t_{p_j}\right| + \left|t_{d_i} - t_{d_j}\right|,$$

where t_{p_i} and t_{d_i} represent the pickup and delivery time for transports i, respectively. A transport is chosen at random and removed, and a predefined number of related transports, according to the measure Δt_{ij}, is removed as well. To be able to find improvements with the use of this removal method, Pisinger and Ropke state that is useful to first select a subset of transports that are geographically close to the randomly chosen transport, before using time-oriented removal. This will make it easier for the algorithm to find feasible changes when working with larger problems. The time-oriented removal method is not included in the software of ORTEC.

Radial Removal

A removal method that is proposed by Schrimpf et al. (2000) is called *radial ruin*. This removal method simply chooses a transport at random, and removes a predefined number of transport that are geographically close to the randomly chosen transport. Just like time-oriented removal takes a single element from the relatedness function from Shaw, this radial removal solely looks at the first element of the relatedness function, d_{ij}. This removal method is known in the ORTEC software under the name related removal.

6.2.2 Number of Removals

Besides the choice of which removal method to use, also choices need to be made about the number of transports that need to be removed. When a small number of transports is removed a large number of solutions can be explored in a short amount of time. However, as stated before in Chapter 4 it is hard to move from one good area of solutions to another only using small changes. Therefore, large moves are needed that can be accomplished by a large amount of removals in each iteration. The disadvantage of this is the increase of computing time, because the number of solutions that is investigated per time unit will decrease [Ropke & Pisinger (2006 nov) or Shaw (1997) or Schrimpf et al (2000)]. In the literature most articles investigate how many transports to remove and whether it suits the problems they look into.

To provide a clear overview on how many transports to remove, Table 6.1 is given. Note that these numbers are found with different test instances. Therefore, one needs to tune the number of removals for the instance that is used. More about these instances in combination with the number of removals is given in Appendix B. It is possible to let

Table 6.1: Number of Removed Transports in Literature

Article	Number of removed transports
Shaw (1997)	around 25%
Schrimpf et al. (2000)	1%, 2%, 5%, 10%, 20%, 50%
Ropke and Pisinger (2006a)	$[4, \min\{100, 0.4n\}]$
Ropke and Pisinger (2006b)	$[\min\{0.1n, 30\}, \min\{0.4n, 60\}]$
Azi et al. (2010)	$[5\%, 35\%]$

The first column in this table shows the articles that is studied from the literature. In these articles, methods like LNS and ALNS are used to find near-optimal solutions for variants of the VRP. Schrimpf et al. (2000) test several numbers of transports that need to be removed, whereas Ropke and Pisinger (2006a), Ropke and Pisinger (2006b) and Azi et al. (2010) choose a random number in the given interval during each iteration. Note that n is the total number of transports in the network that need to be planned on routes.

the number of transports to be removed vary during the search. Shaw (1997) proposes to increase the number of removals, hence, to start with a small number, and to end with many removals. His reasoning behind this, is the fact that he uses a simple construction method for the initial solution, and therefore it should be relatively easy to find improvements at the beginning of the search. However, Ropke and Pisinger state that at the end of the search, their metaheuristic is not accepting many large moves. Hence, they propose to decrease the number of transports that need to be removed, during the search. Note that they use a more complex construction of the initial solution. More about the tests that are performed in the literature on the number of transports that need to be removed, can be found in Appendix B.

6.2.3 Performance

Ropke and Pisinger (2006a) tested related, random and worst removal. They state that for their instances the related removal provided the best solutions, worst removal second best

and random removal provided the worst solutions. From this may be concluded that the two slightly more complicated removal methods perform best. Note that these conclusions are drawn based on the comparison of average gaps between the solutions and the best solutions found during all tests. Unfortunately, Ropke and Pisinger (2006a) do not state the reason for the behavior of the removal methods on their test instances.

Ropke and Pisinger (2006a) also proposed the ALNS, and therefore they are able to use different methods during the same search using a roulette wheel principle, which is explained in Section 4.3. In Ropke and Pisinger (2006b) there are three different configuration that are tested:

- related, worst and random removal with roulette wheel selection,
- related, worst, random, cluster and historical-based removal with roulette wheel selection,
- related, worst, random, cluster and historical-based removal without roulette wheel selection.

The configuration with three removal methods performs worse than the two configurations with more removal methods and the search benefits from the roulette wheel.

Furthermore, Azi et al. (2010) use removal on three different levels: customer level, route level and workday level. They first remove on workday level, random and related removal, followed by removals on route level, random and related removal, and last removals on customer level, only random removal. For each level, the removals are used for a few iterations, before continuing with the next level. Unfortunately, Azi et al. (2010) do not give an explanation why this method performs well on their test instances.

To conclude, which removal method is most suitable will depend on the problem that is considered. It depends on many characteristics, for example the number of transports that needs to be planned and whether there is a restriction on the number of routes that can be made or vehicles that can be used. It will also depend on the method that is used to find the initial solution and on the acceptance criteria that is chosen. For example when a high quality initial solution is used, it would be a waste of this solution to remove a big part of the planned transports. Overall, the heuristic seems to be robust when working with a roulette wheel selection where a few removal methods may be chosen. This method helps to diversify and intensify the search, because some of the removal methods operate on a local level, and others operate on a global level.

6.3 Recreate Methods

A solution is ruined, so when transports are removed, it needs to be recreated by reinserting the removed transports. The number of removal methods is already quite large, but there are even more recreate methods. All algorithms that are known to build solutions can be used as recreate method as well. There are simple insertion methods, such as (parallel) cheapest insertion and parallel regret insertion, but one can also sort the transports before applying the simple reinsertion methods. Examples of these sorting criteria are transportation amount, time window duration and the distance between the pickup and deliver location. With these sorting criteria, one is able to plan the difficult, regarding for example a very tight time window for delivering, transports first.

This section gives an overview of the recreate methods that are used in literature, within the metaheuristics that are studied in this thesis. Note that insertion heuristics in this section are used to schedule unplanned transports on solutions that are already partially constructed. Therefore, the performance of the heuristics may be different than in the case they are used to construct a solution from scratch.

6.3.1 Literature Overview

In the literature, three different recreate methods are used, after removing some of the planned transports. These methods are

- cheapest insertion,
- regret insertion,
- branch-and-bound.

Unfortunately, not all articles elaborate on the reasoning behind choosing a certain recreate method, especially not in the case only one insertion method is used in the metaheuristic. These articles pay more attention to the performance of the removal methods. Moreover, most of the articles studied in this thesis, do not provide the reasoning behind why methods perform well or why they do not.

6.3.2 Performance

Shaw (1997) proposed LNS with the use of branch-and-bound as recreate method. When branch-and-bound is used to reinsert transports, it is able to find a better solution or prove that there is no better solution in a few seconds. This method performs very well for instances with many constraints. The reason Shaw gives is that more constraints imply more constraint propagation, whereby the branch-and-bound procedure can narrow the solution space better. For problems that are very large, and a large number of removals is chosen, it may take branch-and-bound a long time to solve a problem. Shaw noted that the performance of his LNS could be improved by using another recreate method, or even using local search to improve the performance.

Cheapest insertion is used in many experiments, such as in Schrimpf et al. (2000). The reason for Schrimpf et al. (2000) to use this simple recreate method, is to show in what way the metaheuristic works in it most simple form.

Ropke and Pisinger (2006a, 2006b, 2007) test cheapest insertion, regret-2, regret-3, regret-4 and regret-m, where m is the amount of vehicles used in the instance. They tested these recreate methods within the LNS framework, where only one removal and one recreate method is used, and within the ALNS framework, where several removal and recreate methods may be used. The LNS and ALNS heuristics they tested have almost the same performance, when varying the recreate method for LNS. However, the ALNS heuristic is able to adapt to the problem that is looked into, because of the roulette wheel selection, which provides a learning mechanism within the metaheuristic. Therefore, according to Ropke and Pisinger, ALNS should be able to adapt to the instance that is considered, and will therefore be more robust than LNS. Moreover, Ropke and Pisinger state that cheapest insertion, as recreate method, performs worse within their ALNS framework than regret insertion. The quality of the solution is higher with a more complex reinsertion method. However, this comes with the price of a longer computing time.[1]

Azi et al. (2010) use two insertion heuristics, cheapest insertion and regret insertion. Unfortunately, they do not mention the performance of the insertion heuristics within their ALNS framework.

6.4 Acceptance Criteria

In case metaheuristics are used that ruin and recreate a feasible solution, a new feasible solution and the current solution are compared. This newly found solution could be worse or better than the current solution. It could be beneficial to accept the newly found solution in both cases. Up to which criterion a new feasible solution is accepted, is determined by the choice of an *acceptance criterion*. In case a simple acceptance criterion is chosen, it can be very difficult to get away from a local minimum.

To make sure the definitions are understood, we elaborate on the different solutions that are mentioned in this chapter. In the literature, there are three kinds of solutions:

[1]Note that cheapest insertion, when used as a construction method for the initial solution, performs worse than regret insertion, see Chapter 6.1. In case cheapest insertion and regret insertion are used to reinsert transports in an already constructed network of planned transports, the difference in performance is very small.

- the best solution found so far, we refer to it as s^*,
- the current (accepted) solution, referred to as z,
- the new found feasible solution, referred to as s.

Furthermore, we define s_i as the solution found in the i^{th} iteration. Note that $f(s)$ denotes the objective value for solution s, and that we consider a minimization problem. For the best solution so far, found in the i^{th} iteration, the following holds

$$f(s_i^*) \leq f(s_j), \quad \forall 0 < j \leq i.$$

Furthermore, we may accept solutions that are worse than the best solution found so far, hence we may have $f(z) \geq f(s^*)$ at some point in the search. So, it is important to note that the current solution may be worse than the best solution found so far.

Because there are several methods for accepting a newly found solution, an overview is given next, followed by the performance of the accepting criteria that are discussed in the literature. For all acceptance criteria, we included the probability of accepting a new found solution, so that it will provide a way to see the differences and similarities between the acceptance criteria.

6.4.1 Literature Overview

Schrimpf et al. (2000) mention several acceptance criteria, or decision rules as they call them. They do not elaborate on these criteria, they just mention how to use them. In this section, first an overview of these acceptance criteria is given. To show in what way the acceptance criteria are related, we give for each criterion the probability of accepting a newly found solution. As far as the authors knowledge goes, this is not included in the literature, but it will help to understand the differences and similarities between all acceptance criteria. To do so, we use the *indicator function* $\mathbb{1}_A(x)$, where A is an event that depends on variable x. Furthermore,

$$\mathbb{1}_A(x) = \begin{cases} 1, & x \in A, \\ 0, & \text{otherwise.} \end{cases}$$

- The first criterion that is mentioned by Schrimpf et al. (2000) is *random walk acceptance*. With this acceptance criterion all newly found solutions are accepted, even the ones that are worse than the best solution found so far. Therefore, the probability of accepting a newly found solution is equal to 1.
- The second acceptance criterion that looks at worse solutions, besides all solutions that are better, is *threshold acceptance*. Here, a threshold or benchmark T is chosen and solutions with an objective value that lies above this threshold are not accepted. Therefore, solutions are only accepted up to a certain level. Remember that the VRP is a minimization problem, and hence the threshold gives an upper bound for accepting solutions. With this threshold acceptance, only solutions that are 'not much worse' than the current solution are accepted. Therefore, we can define

$$\mathbb{P}(z := s) = \mathbb{1}_{\{f(s) \leq T f(z)\}}(s).$$

- For *greedy acceptance*, a threshold is set to the current solution, hence only solutions that are better are accepted. Hence,

$$\mathbb{P}(z := s) = \mathbb{1}_{\{f(s) \leq f(z)\}}(s).$$

- Another criterion that also accepts worse solutions than the best solution found so far, is *simulated annealing acceptance*. Recall the details on simulated annealing from Section 4.2. The solutions that have a lower value for the objective function than the value for the current solution, are always accepted. However, solutions that are worse

than the current solution, are only accepted with a certain probability that uses a temperature t. This probability is defined by

$$P(z := s) = \begin{cases} \exp\left(-\frac{1}{t}\left(f(s) - f(z)\right)\right) & , f(s) > f(z), \\ 1 & , \text{otherwise}, \end{cases}$$
$$= \exp\left(-\frac{1}{t}\mathbb{1}_{f(s)>f(z)}(s) \cdot \left(f(s) - f(z)\right)\right).$$

- The last criterion mentioned by Schrimpf et al. (2000) is the *great deluge algorithm*. This criterion rejects solutions that are above a certain level, just as in threshold acceptance. However, for this criterion it is more common to set the level in such a way that only solutions that have a very low objective value are accepted.

From this we can conclude that for random walk, greedy en threshold acceptance, the probability of accepting a newly found solution s may be written as

$$\mathbb{P}(z := s) = \mathbb{1}_{\{f(s) \leq T f(z)\}}(s), \tag{6.2}$$

with T sufficiently large for random walk acceptance, $T = 1$ for greedy acceptance and $T \geq 1$ for threshold acceptance. We can also include the great deluge algorithm, as described before, by having $T \leq 1$ in (6.2).

6.4.2 Performance

Schrimpf et al. (2000) compare random walk acceptance with greedy acceptance. They conclude that in nearly all cases with greedy acceptance, the final solution is better than in case a random walk acceptance is used. For a small number of removals, the difference between greedy and random walk acceptance is very small. This is due to the fact that in case only a few transports are removed, the cheapest insertion method will find good solutions rather easily. The removal method does not have a big influence on the performance of random walk and greedy acceptance. They also let the algorithm run for more than 1,000,000 iterations, for both greedy and random walk acceptance. From here can be concluded that for greedy acceptance, the value of the objective function decreases with the increase of the number of iterations, whereas for random walk acceptance, this may not be the case.

Besides the comparison between random walk and greedy acceptance, Schrimpf et al. (2000) also provide tests with threshold acceptance. They use two different cooling schedules, which integrates the idea behind simulated annealing with threshold acceptance. The first is a linear decay for the threshold, and the second one uses a exponential decay. The deviation from the solution and the optimal solution, which is known in this case, decrease for all configurations over time. Moreover, the solutions produced with threshold acceptance are of better quality than the solutions produced with greedy acceptance. Overall, threshold acceptance with exponential decay for the threshold performed best.

Shaw (1997) uses the greedy acceptance criterion. He only accepts new solutions in case they are better than the current solution. Because only one acceptance criterion is used, he does not elaborate on the performance of the algorithm regarding accepting new solutions.

Ropke and Pisinger (2006a) state that simple decision rules have the disadvantage of getting trapped at local minima very often. Worse solutions need to be accepted as well, this makes it easier to get out of these local minima. This is also what Schrimpf et al. (2000) discover with their tests, although they do not state this in their conclusions. Ropke and Pisinger (2006a) propose to use simulated annealing acceptance. With this criterion, they are able to accept many worse solutions at the beginning of the search, and towards the end only accept better solutions. The reason for them to use simulated annealing acceptance, is because at the beginning of the search they want to see whether they can end up at better local minima. Towards the end of the search they hope to be in a good neighborhood where the solutions are very good. Randomization is needed to diversify the search, as is stated

before, but in the ALNS heuristic they propose, this is already covered by the removal and recreate choices. Therefore, randomization is not needed in the acceptance criteria.

Ropke and Pisinger also tried other acceptance criteria, including tabu search algorithms, but this could not provide the same quality for the solutions as simulated annealing acceptance.

Chapter 7

Testing Approach

Recall the ORTEC ALNS that we will test in this thesis from Algorithm 4. As explained in the introduction of this thesis, given in Chapter 1, the focus of this thesis is on how to configure ALNS and its components such that it can find near-optimal solutions for a rich and real-world VRP. Moreover, in Chapter 3 it is explained that we do not know the optimal solutions for the instances that are tested on, because we work with real-world data. Therefore, it is hard to know whether the solutions that are found by our ALNS configuration are near-optimal.

This chapter is included to explain our testing approach. In Section 7.1 we elaborate on the approach for tests that are executed on the components of our hybrid method ALNS. Furthermore, Section 7.2 describes in what way the solutions will be compared. The tests are performed on two cases, which we will elaborate on in Section 7.3. We chose to include the data in this chapter, to be able to have one chapter, Chapter 8, that only focuses on the experiments that are executed and the results.

7.1 Components of ORTEC ALNS

Chapter 6 provided an overview of the components that are tested. When these components are put together into one heuristic, this gives our ORTEC ALNS. The performance of each component depends on the performance of the other components. All components have several options and parameter settings that need to be tuned. Testing every combination that is possible for all components, including all parameters, would take a very long time. Therefore, we need to make choices about the testing approach.

- First, initial tests are executed. These tests are included to see in what way the instances react to the hybrid method. For example, the computing time for removing and recreating solutions will influence the possible number of iterations. The initial tests will be based on the findings from the literature and combined will form several initial configurations. What local search methods and which acceptance criterion need to be used are tested as well. These components of the hybrid method are not tested separately, but we recommend to include tests in further research. What configurations are chosen from these initial tests, will be based on the best values for the objective function.
- The second step is the tuning of each component of the hybrid method, while the other components remain fixed. The following tests will be executed.

    1. **Construction methods for the initial solution.** Several construction methods are tested, including simple heuristics such as parallel regret insertion, but also several versions of sequential and parallel insertion. These tests are first executed without the ALNS framework, to see what method is able to plan all transports. Tests including the ALNS will follow next. The most promising

construction methods for the initial solution will be used in further experiments for the other components.

2. **Removal methods.** The experiments that are performed on the removal methods are split into two categories: multiple removal methods, with and without roulette wheel, and single removal methods. The removal method that provided the best results is used for tests that are performed on the number of transports that need to be removed. These tests are executed to see the influence of one percentage of transports that is removed during the search.
3. **Recreate Methods.** To see the effect of the recreate method as component of the hybrid method, several recreate methods will be included as single method for reinserting removed transports. That is, so far several recreate methods are used with a roulette wheel, but now we also test the configuration with one recreate method. Tests are executed on configurations with two methods for constructing the initial solution and with one removal strategy from both categories (multiple and single removal methods).

- From the configurations that are tested in the second phase, two configurations are chosen to perform final tests on. One will be the configuration with the best objective function value, this will be our hybrid ALNS configuration. A second configuration will be chosen that is more simple, with only one removal and one recreate method that are used during the search. Final tuning on the number of transports that need to be removed during each iteration will be done.

Recall from the previous chapter that the acceptance criterion is also seen as a component. We will only test on this component in the initial tests, but further research can be done according to this acceptance criterion.

7.2 Solution Quality

We will elaborate on the method that is used to measure the solution quality and in what way the solutions that are found with different configurations are compared.

As explained in Chapter 3, the objective function in the software of ORTEC has a certain hierarchy. Therefore, the solution will not provide one objective function value, but there are several key performance indicators, KPIs, that can be compared. The plan costs are the first main objective and therefore the solutions will be compared based on this KPI. However, the other KPIs are mentioned as well during the tests, especially when the plan costs of solutions are very close or in case some KPIs are very low or high.

For all methods that are tested in the components, the average KPIs over all instances will be given in tables. The computing time for the configurations is also included, because usually good performing methods come with the price of a long computing time. Next to tables, also figures are included as a visualization method for comparing the plan costs. For all components we also split the average KPIs into averages for the two construction methods for the initial solution. Tables can be found in Appendix D and figures are included in the next chapter. These are included to see whether the construction method for the initial solution has influence on the performance of the components of the hybrid method. This is not yet included in the literature that is studied for this thesis.

Furthermore, the final configurations of our ALNS can be compared with the original customer configurations. These configurations are the ones that are used at the customer to plan the transports on the routes.

7.3 Data for Testing

For the tuning of the hybrid method, three different instances are used from one customer of ORTEC. These instances are referred to as *training instances*. Furthermore, 14 instances from another customer are used to verify the conclusions that are drawn based on the

training instances. We refer to them as *test instances*. The training instances are referred to as B1, B3 and B4 and the test instances are named C36 up till C49 The instances are named based on files that were obtained from the customers. Because the customers did not provide data from consecutive days, the instances do not have consecutive numbers. In order to prevent misunderstandings and lost data, we chose to keep the naming of the data as it is provided by the customer.

Table 7.1: Data used for testing

Instance	Number of				
	addresses	transports	tasks	routes	transports per route
B1	156	397	794	49	8.10
B3	124	397	794	56	7.09
B4	91	128	256	41	3.12
B total	371	922	1844	146	6.32
C36	-	221	442	137	1.61
C37	-	389	778	188	2.07
C38	-	245	490	105	2.33
C39	-	346	692	163	2.12
C40	-	209	418	140	1.49
C41	-	384	768	190	2.02
C42	-	238	476	100	2.38
C43	-	344	688	187	1.84
C44	-	211	422	139	1.52
C45	-	387	774	200	1.94
C46	-	233	466	109	2.14
C47	-	359	718	188	1.91
C48	-	214	428	144	1.49
C49	-	368	736	195	1.89
C total	-	4148	8296	2185	1.90

An overview of some of the characteristics from the data that is used for testing. The first column gives the name of the instance, for two sets of testing: B for the training instances, C for the test instances. The other five columns show the number of addresses, transports, tasks (twice the number of transports), routes and the transports per route. The addresses are the customer locations for delivery, which are not stated for the test instances C due to customer protection. The routes that are stated are the number of vehicles that can be used for planning, where the vehicles have empty initial routes.

Because there are many configurations tested, it would be time consuming to test them all on both the training and test set. Most of these configurations run for hours. Therefore, the training set is small, to perform many tests with different configurations. The initial tests are not performed on the test set, but only the configurations per component are executed on the test set. The number of iterations is less than for the training set, to speed up the process of testing.

An overview of some of the variables for the instances is given in Table 7.1. The main difference between the two sets of instances, that can be seen in this table, are the number of transports that are planned per route. For the instances B this is a little over six, on average, and for the instances C almost two.

Furthermore, note that the transports per route vary more for the instances B than for the instances C. The difference between the instances B is caused by the size of the transports, hence the number of pallets that are needed per transport. For B4, the amount

of pallets is on average higher per transport than for the other two instances. This can be seen from the number of transports per route, as the vehicles can load less transports per vehicle for instance B4. Moreover, the transports per address are less for B4 than for the other two instances of B.

All instances have one depot and multiple products are transported that come with their own restrictions. Some products can not be transported together, other products need to be delivered with a certain type of vehicle. Therefore, also several types of vehicles can be used in the planning, with different capacity and characteristics. Hence, the fleet is heterogeneous. All transports have time window restrictions for delivery.

Chapter 8

Experiments

This chapter describes the results of our experiments, according to the testing approach described in the previous chapter. First of all, the original customer configurations are reported on, to see whether all the transports are planned on the routes. An overview is given in Section 8.1. Next, initial tests are performed and reported on in Section 8.2, which will result in three initial configurations that will be used in the experiments per component. Recall from the previous chapter the components that will be tuned:

- Initial Solution, Section 8.3,
- Removal Methods, Section 8.4,
- Recreate Methods, Section 8.5.

Each component that is tested on is reported in the same way: first the results of the training instances B are given and discussed, followed by the results of the test instances C. To avoid confusion, the colors in the figures for instances C are different than the colors of instances B. A conclusion, based on both the training and test instances, is provided at the end of each section that will help answering the subquestions of this thesis.

After the experiments on all the components are discussed, two configurations will be constructed based on the results of the tests per component, in Section 8.6. Some additional tests will be reported on, including tuning the number of transports that need to be removed.

Furthermore, recall from the previous chapter that the instances are not all consecutively numbered, but the configurations are. Not all these configurations are mentioned in this thesis, as they did not all provide useful outcomes. Therefore, the numbering of configurations in tables might skip a few numbers. The configurations in this report and the research are kept the same, to prevent confusion and to be able to re-use configurations after reading this report.

All experiments were performed on a Windows 7 based PC, equipped with 4GB RAM and two 2.60GHz processors.

8.1 Customer Configurations

As already explained in Chapter 7, we are not able to compare our results with the optimal solution, because we do not know the optimal solution for these real customer cases. To have an idea of the performance of the hybrid method that we test, the solutions constructed with the original configuration from the customer are seen as upper bounds. That is, at least we want to have lower KPIs than with this configuration, given in Table 8.1. In previous versions of the ORTEC software, a different cost set was used at the customer. The costs per hour, a KPI that is used for optimization, was set 100 times the real value, in order to reduce the wait time in the solution found during optimization runs. The wait time is included in the KPI hour, along with, for example, driving time and handling times at the depot and location. In case the costs per hour are set higher, it will be more expensive

to wait, for example, at a location till a time window of delivering opens. With these extra costs per hour, one can steer the optimization process in a direction where the wait time is lower. However, we also test with the real costs per hour, included in Table 8.1, to see whether the new version of the ORTEC software is able to decrease the wait time, without altering the real costs. Furthermore, as the instances C are used as test instances, and the main goal is to see whether the components behave in the same way for these instances, we do not include the customer configurations for case C. From Table 8.1 can

Table 8.1: Original Customer Configuration (Average)

Configuration	Case	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
Original	$B1_c$	41	794	937,161	5576	185.50	340,510	02:47
Original	B1	40	794	18,842	5476	187.32	337,600	04:20
Original	$B3_c$	52	790	1,054,746	6793	208.17	390,189	03:53
Original	B3	50	790	23,612	6836	197.53	396,963	02:57
Original	$B4_c$	32	256	606,054	4155	119.49	248,476	00:32
Original	B4	31	256	14,287	4073	117.49	247,793	00:28
Original	All_c	125	1840	865,987	5508	171.05	326,392	02:24
Original	All	121	1840	18,914	5462	167.45	327,452	02:35

The small c in the column 'case' is used as an indication of different configured cost sets. For cases with the small c, the costs per hour are set 100 times the real costs per hour, in order to decrease the wait time in the solution. The columns trips and tasks give the number of trips and tasks that are used and planned. The plan costs, in euros, distance, in kilometers, hours and driving time, in seconds, are KPIs to indicate the performance of the configuration. The last row shows the sum of all trips and tasks from the previous rows and the average of all other KPIs.

be seen that the number of tasks, so twice the number of transports, that is planned with a new cost set is the same as with the original configuration from the customer, denoted with a small c for the column 'case'. Therefore, we are able to compare the solutions in a fair way. With the new version of the software of ORTEC, we see that it no longer seems to be useful to alther the cost set in order to reduce the wait time. When comparing the different cost configurations for case B3, we see that the driving time increased, but the total hours decreased. Hence, the wait time in the solution did decrease. For case B4, the driving time decreased with around 11 minutes when the costs are set equal to the real costs. However, the hours decreased by two. Therefore, we may conclude that the wait time in this solution did decrease compared to the configuration with the higher cost set. However, for B1 the hours did increase, whereas the driving time decreased. So indeed, we may conclude that for this case, the wait time increases in case the costs per hour are set to the real costs. Moreover, for all cases we see that the number of trips is less with the real cost set. Therefore, we will continue to work with the real cost set instead of the original cost set used for the customer.

8.2 Initial Testing

As mentioned in Chapter 7, initial tests are performed to see in what way the instances react to the hybrid method. These initial tests are performed on instance B1. Note that the results for the initial tests can not be used to draw conclusions on the methods and numbers that are tested. Testing on one instance will not provide enough information about the performance of the method. However, as explained in Chapter 7, these initial tests are executed to have a starting point for our hybrid method that is not solely based on intuition and the literature. After all initial tests are executed on B1, the most promising

configurations will also be tested on B3 and B4. With the conclusions that are drawn from these tests, three configurations will be chosen that serve as starting points for our tuning per component of the hybrid method. A few steps are followed to come up with an initial configuration:

- **Plan all transports.** To be able to compare the configurations, the number of transports that is planned per case needs to be the same. Otherwise, it is not 'fair' to compare the KPIs. For example, the total distance that is driven in the network usually increases with the number of transports that are planned. To do so, we tested three insertion methods: parallel cheapest insertion (PCI), parallel regret insertion (PRI) and cheapest insertion (CI). For all insertion methods, four different estimators are tested: driving time, wait time and costs, plan costs and distance. Initially, these simple insertion methods were not able to plan the same number of transports as the original customer configurations. Therefore, 100 iterations of local search, a 2Opt, are executed and the remaining transports are planned. The only methods that eventually were able to plan the same number of transports as the original customer configurations are PRI and PCI based on distance.
- **Removal Methods.** For the initial tests, five different removal methods are used: related, random, worst, cluster random and cluster worst removal. These methods remain fixed during the initial tests and are tuned during the tests per component. These five methods are used with a roulette wheel selection. This choice is based on the literature, a learning layer and multiple removal methods seems to provide good configurations (Ropke & Pisinger, 2006b).
- **Recreate objective.** The recreate methods are the same as with the construction for the initial solution: PCI, PRI and CI. Again, four different estimators are tested: driving time, wait time and costs, plan costs and distance. We found that reinserting transports based on distance results in the lowest plan costs for the overall solution for B1.
- **Number of iterations.** To get an idea on how many iterations may be used for the hybrid method, several tests are performed. For a total of 12,000 iterations, the software of ORTEC still performs good. The calculation time does not exceed 20 hours and the maximal physical memory is not exceeded. These 12,000 iterations are split up into three parts, according to Ropke and Pisinger's number of removals (Pisinger & Ropke, 2007). They advise to remove a number of transports that lays within $\left[\min\{0.1n, 30\}, \min\{0.4n, 60\}\right]$, where n is the total number of transports in the network. Because B1 has 397 transports to plan, we have a minimum of 30 and a maximum of 60 transports that need to be removed. In the software that we use, we are not able to take a random number within an interval and remove this random number of transports. The number is a parameter that needs to be specified upfront. Therefore, we start our tests with 30, 45 and 60 transports that need to be removed during one run of the hybrid method. Each number is used for a predefined number of iterations before the next amount is used. Because 12,000 iterations are required in total, we test 16 x 250, 8 x 500, 4 x 1000 and 2 x 2000 iterations. For example, with 16 x 250, the hybrid method removes 250 times 30 transports, followed by 250 times 45 and 250 times 60 transports. This results in 750 iterations for one loop over the outer framework. A recursion of 16 over these 750 iterations results in a total of 12,000 iterations. From these tests, 2 x 2000 iterations came out as best option, with the lowest plan costs for B1.
- **Number of removals.** Because Shaw (1997) advocates decreasing the number of removals, opposed to Ropke and Pisinger (2006a), we also test 60, 45 and 30 transports to be removed. However, this did not provide us with a solution with lower KPIs. Furthermore, within ORD, the software of ORTEC, the number to be removed is specified as the number of tasks. Therefore, to remove one transport, two tasks need to be removed. Hence, we also test the double amount: 60, 90, 120 and 120, 90, 60. The solutions found with both methods had higher KPIs than the one with 30, 45

and 60 transports that had to be removed. Furthermore, the initial tests included a wider range of removal iterations, namely: 30, 50, 70, 90, 110, 130 (and in reverse order). The solution found with 30, $\cdots$, 130 was very close to the solution found with 30, 45 and 60 transports to be removed.

- **Local Search.** After each step of removing transports and recreating the solution, local search is included to improve the newly found solution. However, we only execute the local search in case the newly found solution is at most 5% worse than the best solution found so far. Hence, we set the threshold to 1.05 for executing the local search. This local search involves 100 iterations of two local search methods: 2Opt and Move. Recall their definition from Chapter 5. We also tested with more local search methods during each iteration. The results regarding the plan costs improved, but the calculation time significantly increased (around a factor of six). Because the configuration is tested thoroughly, we choose to only use the quick local search methods 2Opt and Move to control the running time.

  Furthermore, tests were executed with a threshold of 1.01. However, the solutions that were found with this threshold had higher KPIs, and thus a worse quality. More diversification, with temporarily accepting new solutions, is included with a threshold of 1.05, which helps the heuristic in the search for near-optimal solutions.

- **Alternative Local Search.** Instead of using local search to improve the newly found solution, in case it is at most 5% worse than the best found solution so far, we also tested with another round of removing and recreating the new found solution. Hence, in case the newly found solution is within the threshold that is specified, 10 iterations of removing 30 tasks and recreating the solution takes place. This includes a roulette wheel, where three removal methods are used (related, random and worst removal) and the removed transports are reinserted with cheapest insertion (based on distance). In case the newly found solution is within 1% of the best found solution so far, local search is included (2Opt, move).

  The solution found with this method has slightly lower KPIs than the one that was found with 'normal' local search. Especially the hours did decrease.

These initial tests are only performed on one case. The most promising and interesting configurations are tested on the other two cases as well. The choice for this subset of configurations is based on the literature and on conclusions from the tests performed on instance B1. These tests are executed to see whether all transports are planned, and to get an idea of the performance of these configurations on other instances. An overview of which components are included in these configurations is given in Table 8.2. Table 8.3 gives an overview of the results of the configurations that are tested on the three training instances. First of all, note that the routes and tasks are summed up over all instances, all other KPIs are averages. We can motivate summing up the routes and tasks by the fact that averages may result in fractional routes or tasks. This is the result of having several instances.

From the first three rows in the table can be seen that only PRI based on distance is able to plan the same number of transports as the original customer configurations. Furthermore, adding local search to improve the new found solution after one removal and one recreate step improves all KPIs. To see this, compare configuration 78_a with all configurations up to 109_a. It is striking that the total number of routes that is used for the three cases does not decrease when local search is used. This indicates that the planning is more effective when it comes to which orders are planned on which routes. Especially the decrease in hours stands out, which is more than 7.5% for all other configurations compared to the configuration without local search.

Furthermore, we tested with several numbers of transports that need to be removed, which can be seen in the third column of Table 8.2. Not only the amount to be removed varies, but also whether more or less number of transports are removed during the search. For our research, we want to use two different configurations when it comes to the amount to remove, but also whether this is in a increasing or decreasing fashion. Considering the results

Table 8.2: Configurations of Initial Tests

Configuration	Initial Solution	Removals	Local Search	Acceptance Criterion
69	PCI (d)	30, 45, 60	-	-
72	PRI (d)	30, 45, 60	-	-
73	CI (d)	30, 45, 60	-	-
78a	PRI (d)	30, 45, 60	-	-
86a	PRI (d)	60, 90, 120	2Opt/Move	Th 1.05
$86a_r$	PRI (d)	120, 90, 60	2Opt/Move	Th 1.05
93a	PRI (d)	30, 45, 60	2Opt/Move	Th 1.05
$93a_r$	PRI (d)	60, 45, 30	2Opt/Move	Th 1.05
97a	PRI (d)	30, 50, 70, 90, 110, 130	2Opt/Move	Th 1.05
109a	PRI (d)	130, 110, 90, 70, 50, 30	2Opt/Move	Th 1.05
117a	PRI (d)	30, 45, 60	Alternative	Th 1.05
122a	PRI (d)	30, 45, 60	Alternative	Th 1.01

An overview of the configurations that are executed for the initial tests. From left to right: the configuration name, the construction for the initial method with the objective to insert the transports (distance): parallel cheapest insertion (PCI), parallel regret insertion (PRI) or cheapest insertion (CI), the number of transports that are removed, the local search method and the threshold that is used for either local search or the alternative local search as explained in the initial tests. Furthermore, between brackets in the column 'Initial Solution' is the estimator that is used to insert transports. For these configurations, distance (d) is used. Also, in the column 'Local Search', alternative is explained in the initial testing steps.

in Table 8.3, we see that configuration 109a found on average the solution with the lowest plan costs, distance and driving time. The hours are slightly higher than with configuration 86a. This can be explained by the diversification in the search. This configuration has less iterations per fixed number of transports that are removed, before moving on to the next number. Therefore, more diversification is included than for example with 30, 40 and 60 removals. Hence, 109a will be the configuration that is chosen for the tests per component where the number of transports that are removed are of decreasing order.

We also need a configuration that has an increasing order of transports that are removed during the search, to keep the results of the next experiments general. Therefore, we can either choose configuration 86a or 93a for the tests that will follow. To make sure the tests that are performed per component are as general as possible, we chose to work with the configuration that results in slightly higher KPIs. Hence, we will use configuration 93a as initial framework, as the combination of the components that are chosen at the end of this research need to fit in configurations that produce slightly higher KPIs as well.

The last configuration that is included as starting point for the experiments per component of the hybrid method, will be 122a from the last row of Table 8.3. We chose to use this configuration, because of the alternative way of local search. Instead of simple improvement heuristics such as 2Opt and Move, we include extra diversification by applying an extra set of fast ruin and recreate iterations. As already explained, 30 iterations are used: 10 iterations per three removal methods, where only 30 tasks are removed. As expected, the computing time for this configuration almost doubled with respect to the other configurations that include local search. To summarize, the configurations that are used as starting point for the tests per component are 93a, 109a and 122a. From now on, we will refer to them as configurations 93, 109 and 122 respectively. Note that the upcoming tests show results that are averages over these three configurations. As explained in Chapter 7, the performance of the components can be influenced by the rest of the configuration. Therefore, the performance of the components is measured and evaluated as average over

Table 8.3: Results Configurations of Initial Testing (Average)

Configuration	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (h:min)
69	117	1814	19,493	5562	178.83	337,297	4:48
72	121	1840	19,559	5644	176.04	338,997	5:35
73	119	1826	19,752	5697	179.74	344,925	4:18
78a	122	1840	19,534	5520	177.42	331,900	2:29
86a	122	1840	18,372	5213	161.03	313,833	3:17
$86a_r$	123	1840	18,484	5160	163.54	313,013	2:31
93a	123	1840	18,567	5194	164.26	313,582	2:09
$93a_r$	123	1840	18,481	5163	163.43	313,540	2:26
97a	122	1840	18,398	5218	161.48	314,842	2:20
109a	122	1840	18,313	5134	161.26	311,244	2:34
117a	122	1840	18,393	5164	162.18	311,662	4:18
122a	122	1840	18,340	5138	161.57	310,333	4:30

The first column contains the number of the configuration, or template, that is executed. These names correspond to the configurations in Table 8.2. Furthermore, the trips and tasks are summed up over all three training instances, which in total should lead to 1840 tasks that are planned. The other columns contain KPI values to indicate the performance of the configurations, which are averages over all training instances. There are three different components in this table, divided by horizontal lines. From top to bottom: three configurations for initial tests on the construction method for the initial solution, seven configurations for tests on the number of tasks that need to be removed and the local search method, and two configurations with an alternative local search method.

the three different configurations of ALNS.

8.3 Initial Solution

The first component of our hybrid method that is tested, is the construction of the initial solution. From the initial tests that were described in the previous section, we found that for a simple construction method, parallel regret insertion based on distance plans the same number of transports are with the original customer configuration, with the lowest plan costs. However, we have more influence on the construction of the routes with sequential and parallel insertion, where we can order the transports for insertion. These commonly known insertion methods from the literature, have many variants. For example, seeds may be chosen in different ways, so the initialization of the routes can be done in several ways. Therefore, we tested three different sequential insertion methods and two parallel insertion methods. Examples of one sequential insertion algorithm and one parallel insertion algorithm are given in Algorithms 5 and 6 respectively. The three sequential insertion algorithms only vary in line 3, where the sorting keys are used, and the two parallel insertion algorithms vary in line 5. The delivery tasks are sorted in different ways, with the help of sorting keys, to guide the planning. For the five algorithms, the following sorting is used for the tasks

- farthest away from the depot with the largest quantity for pickup and delivery (145),
- farthest away from the depot with the latest delivery time window possibility and the biggest quantity for pickup and delivery (145a),
- the latest delivery time window possibility for the task that is farthest away from the depot with the biggest quantity for pickup and delivery (145b),
- closest to the already planned addresses with the biggest quantity for pickup and delivery (146), and

- closest to the already planned addresses (146a).

Algorithm 5: Sequential Insertion (VRPO specific)

```
   begin
1      Sort all routes (vehicles) based on the capacity, in decreasing order
2      for All routes (vehicles) do
3          Order the delivery tasks based on the sorting keys, construct a list S
4          Plan a seed tasks, with cheapest insertion, that is the first entry in S
5          Order the delivery tasks based on the smallest distance to the address of the
             seed task, construct a list L
6          Plan tasks, with cheapest insertion, starting with the first entry in L, till the
             route is full
       end for
7      Perform local search to improve the solution
8      Try to insert unplanned tasks
   end
```

Algorithm 6: Parallel Insertion (VRPO specific)

```
   begin
1      Sort all routes (vehicles) based on the capacity, in decreasing order
2      Sort all delivery tasks based on the latest delivery time window possibility for the
         transport with the biggest quantity for pickup and delivery, construct a list S
3      for All routes (vehicles) do
4          Plan a seed tasks, with cheapest insertion, that is the first entry in S
       end for
5      Order the delivery tasks based on the sorting keys, construct a list L
6      for All tasks in L do
7          Plan tasks with parallel regret insertion
       end for
8      Perform local search to improve the solution
9      Try to insert unplanned tasks
   end
```

From these tests with several construction methods, we choose one sequential and one parallel insertion method to test with in the next components. The average results over all cases is given in Table 8.4.
Looking at the results of tests on the construction method for the initial solution in Table 8.4, we see that the number of transports that is planned is more than what was planned with the original customer configurations. Furthermore, with PRI, based on distance, we are not able to plan more transports, as we have seen in the results in Table 8.3. Hence, constructing the routes where we influence the order in which the transports are inserted is needed to plan all transports that are given in the network.

For further testing, we will choose one sequential and one parallel insertion method. First consider the parallel insertion methods, for which the results are shown in the last two rows of Table 8.4. Unfortunately, not all transports are planned with configuration 146, hence 146a will be used for further testing. The seeds for the routes are the same for both configurations, hence the difference is in what way the remaining transports are planned. When the transports that have the biggest quantity for pickup and delivery are planned first, after the seeds, the routes will become full due to the capacity constraints of the vehicles quickly. It may happen that smaller transports that are very close to the seed

Table 8.4: Results Initial Solution, without ALNS (Average)

Configuration	Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
145	SI	120	1844	21,295	5968	204.22	354,467	05:17
145a	SI	122	1844	20,706	5816	194.26	345,338	06:38
145b	SI	120	1844	21,325	5972	204.18	355,772	07:14
146	PI	133	1842	20,090	5570	179.93	330,672	03:50
146a	PI	133	1844	20,236	5541	183.08	331,159	02:28

Each row represents the results of a configuration that is tested for the construction of the initial solution, without any removing and recreating of the solution. Tests are executed on three training instances, the average results per configuration are shown. From left to right: the configuration that is used to obtain the result; the method for the initial solution, either sequential insertion (SI) or parallel insertion (PI); the sum of the trips that are used; the sum of the tasks that are planned; the average of other KPIs that are used to indicate the performance of the configuration.

of the routes are kept unplanned till the end of the search. In that case, detours are needed to plan these transports with smaller delivery quantities, which might lead to transports that are left unplanned.

Next, consider the sequential insertion methods that are reported on in Table 8.4. All KPIs for configuration 145a are the lowest, expect for the number of routes that are used. For this configuration, the routes are built backwards, hence the transports that have a delivery that is farthest away from the depot with the latest delivery time window possibility are inserted first. However, we want to test whether the quality of the initial solution is important for the quality of the other components, as this is not mentioned in the literature that is studied for this research. Therefore, we choose to use a sequential insertion method with KPIs that are further from the KPIs of the parallel insertion configuration we chose, which results in using configuration 145 for further testing. Configuration 145 produces a solution with slightly better KPIs than configuration 145b.

Furthermore, note that the KPIs for the solution that is produced with PI are lower than the solutions that are produced with SI. The hours decreased the most, with 5.6% with respect to the best configuration for SI. This is caused by the extra routes that are used, around 9% more with respect to configuration 145a. Because the number of transports that are planned is the same, the routes have, on average, less transports to deliver. Therefore, the routes are built more efficient.

To summarize, further testing will be done with two construction methods for the initial solution:

- SI, where the transports that are farthest away from the depot with the largest quantity for delivery are planned as seed, followed by the transports that are closest to the already planned address.
- PI, where the transports that have the latest delivery time window possibility with the biggest quantity for delivering are planned as seeds, followed by the transports that are closest to the already planned address.

Training Instances B

The previous section provided us with two different configurations for constructing an initial solution: sequential insertion (SI) and parallel insertion (PI). To provide a complete overview, the results of tests that use parallel regret insertion (PRI) are also included. The average results over all training instances, instances B, is given in Table 8.5. Note that the average is taken over three instances and three different configurations (93, 109 and 122),

hence this is the average over nine different configuration and instance combinations. The routes, or trips, and tasks are summations over all these cases.

Table 8.5: Results Initial Solution, with ALNS (Average)

Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (h:min)
SI	363	5532	18,452	5193	163.16	314,578	3:27
PI	372	5532	18,399	5165	160.72	311,563	3:37
PRI	367	5520	18,407	5155	162.36	311,720	3:04

The average results for the tests on the construction for the initial solution, executed on the training instances B. The average is taken over three training instances in combination with three different configurations. This results in nine cases, where the maximum number of tasks to plan is equal to $(3\times794)+(3\times794)+(3\times256) = 5532$. The first column contains information about the method used for constructing the initial solution: sequential insertion (SI), parallel insertion (PI) and parallel regret insertion (PRI). The columns trips and tasks show the sum of all trips and tasks of the nine cases. The last five columns represent all other KPIs that are used to indicate the performance of the configurations.

We see that not all transports are planned with PRI as construction method for the initial solution, what was already mentioned in the previous section. This might be the result of the simplicity of the insertion method. Furthermore, because there is not a seed transport inserted in each route, the regret value will be equal for inserting a transport in any empty route. Therefore, randomness is involved with initializing the routes. However, when looking at Figure 8.1 we see that the plan costs for PRI are not even the smallest. Hence, with less transports being planned, PRI is not the best performing construction method for the initial solution, based on plan costs only.

Furthermore, we notice that all KPIs for SI are slightly worse than for PI, even though PI uses more routes in its solution. Using more routes may imply less detours, which might explain the decrease in distance, hour and driving time. For PI all routes are initialized with a seed transport, after which the transports may be moved to a better route. This can be an explanation for the increase in routes compared to SI, where the routes are constructed one by one.

Although there are some differences in performance when using different construction methods for the initial solution, these differences are very small. The decrease of the plan costs of PI with respect to SI is around 0.28%. However, when considering the other KPIs, especially the hours decreased for PI with respect to SI, namely around 1.49%. This comes with the price of an increase of used routes with 2.48%. Recall that for SI and PI without removing and recreating parts of the solution the differences were bigger: the plan costs for SI were almost 5% higher than the plan costs for PI. This might indicate that removing and recreating parts of the solution will partly make up for the transports that were not inserted in the cheapest place for the initial solution.

Recall that in the literature, the construction of an initial feasible solution is not discussed, as far as we have observed. Most articles assume that such a solution is already known and can be used as input for their algorithms. The minor differences in KPIs when using different construction methods may be the reason for this.

Test Instances C

Tests on the initial solution are executed for the test instances as well. The results are shown in Table 8.6 for the construction method for the initial solution, and in Table 8.7 including ALNS. Note that all transports are planned, because all tasks are planned, with all three initial solution. Regarding the plan costs only, we see in Table 8.6 that SI performs best, followed by PI and PRI. The gap between the plan costs for SI and PI is small, whereas the plan costs for PRI are higher. Considering the other KPIs as well, we notice that SI and

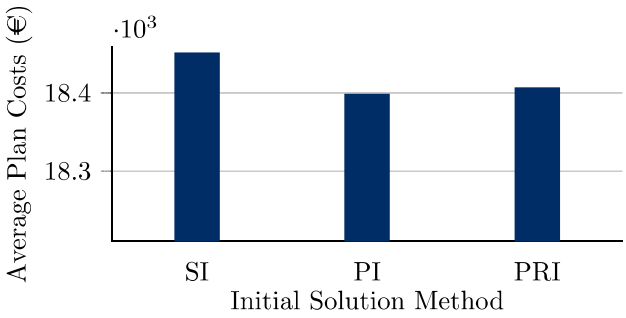


Figure 8.1: The average plan costs, over three configurations and three instances, for three configurations with a different construction method for the initial solution: sequential insertion (SI), parallel insertion (PI) and parallel regret insertion (PRI). The construction of the initial solution is followed by 12,000 iterations of removing and recreating the solution. Note that the y axis is in thousands (euros) and the axis value does not start at zero.

PI have similar values, where SI performs slightly better. PRI produces by far the worst initial solution, although the number of routes that is used for PRI is the same as for PI. This shows the strength of PI as initial solution, because the routes are built in a more clever way than with PRI.

However, when ALNS is included, the results are remarkable. Table 8.7 shows that, regarding the plan costs, constructing the initial solution with PRI and applying ALNS performs the best. Although the gaps for the plan costs between the different initial solution methods, with ALNS, are small, we can notice an influence of the method for constructing the initial solution. Comparing the tables with and without ALNS applied after the initial solution is built, we see that an initial solution with the highest plan costs, PRI, has the lowest plan costs after ALNS is executed. The decrease for all KPIs when including ALNS is around 2% for SI and PI and around 19% for PRI. From this we can conclude that ALNS is able to find more improvements in case the initial solution is further away from the optimal solution.

Furthermore, we see that PRI uses less routes to plan all transports and the total distance is less than for SI and PI. Nevertheless, the total hours are slightly higher with PRI than SI and PI, from which we can conclude that the wait time is higher for PRI. This can be explained by the fact that with less routes, and the same number of transports, the number of transports per route is higher for PRI. The wait time can be explained with the time window restrictions on the transports. For example, route A starts at 6 am and has its first delivery at 06:30. Including an extra transport with a time window delivery from 4 till 5 am, causes the start time of the route to be scheduled earlier. The first delivery will move to be the second delivery in the route. After delivering the first transport, a wait time will be included till the time window of the second delivery opens.

Conclusion

Taking all performance results into account, PRI will no longer be included in the upcoming tests for the other components of the hybrid method. This is a consequence of the transports that were left unplanned for some of the test instances. We will include both SI and PI as construction method for the initial solution, resulting in six different configurations for the next components that are tested in combination with the configurations 93, 109 and 122. Regarding all KPIs, PI performs best as construction method for the initial solution in combination with ALNS. Note that the difference in performance between SI and PI is very small, although the difference in performance was bigger without removing and recreating parts of the solution for some of the KPIs. Furthermore, for the test instances, we can

Table 8.6: Results Initial Solution C, without ALNS (Average)

Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
SI	2226	8296	5,669,225	52,360	2007.13	3,759,494	01:42
PI	2231	8296	5,678,027	52,436	2012.96	3,763,333	01:05
PRI	2231	8296	6,895,190	63,879	2377.02	4,570,004	00:42

The average results for the tests on the construction method for the initial solution, without removing and reinserting transports, for the test instances C. The averages are taken over all 14 instances, which results in a total sum of maximum to plan tasks of 8296. Each row represents the results for one construction methods: sequential insertion (SI), parallel insertion (PI) and parallel regret insertion (PRI).

Table 8.7: Results Initial Solution C, with ALNS (Average)

Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
SI	6561	24,888	5,558,084	51,344	1962.10	3,692,100	48:36
PI	6556	24,888	5,557,866	51,327	1969.87	3,690,645	49:07
PRI	6548	24,888	5,552,740	51,276	1970.00	3,686,559	52:38

Each row shows the average results of a construction method that is used for the initial solution, followed by removing and reinserting transports, for the test instances C. The average is taken over 14 instances and three initial frameworks for ALNS, which results in a total of $3 \times 8296 = 24,888$ tasks that need to be planned. Note that these configurations included a total of 1200 iterations of ruining and recreating the solution.

conclude that an initial solution that is of poor quality can produce high quality solutions after ALNS is applied. Although this is not seen in the training instances, it is worth mentioning.

8.4 Removal Methods

The second component of the hybrid method that will be tested is the removal method. From the literature we have seen that there are several ways of removing a part of a feasible solution, after which it is reconstructed again. Shaw (1997) uses only related removal, whereas Ropke and Pisinger (2006b) use multiple removal methods within their ALNS method. For ORTEC, there are six different removal methods that can be executed: related (SR), random (RR), worst (WR), cluster random (CRR), cluster worst (CWR) and trip removal (TR), as described in Section 6.2. There are many configurations that can be tested, because there are many combinations of the removal methods. Furthermore, inspired by Ropke and Pisinger (2006b), the learning layer is tested as well in the form of a roulette wheel, as described in Section 4.3. Hence, the following configurations will be tested

- three removal methods (related, random and worst removal) with and without roulette wheel, denoted by 3RW and 3noRW,
- five removal methods (cluster random and cluster worst removal are added) with and without roulette wheel, denoted by 5RW and 5noRW,
- six removal methods (trip removal is added) with roulette wheel, denoted by 6RW,
- five single removal methods (related, random, worst, cluster random and cluster worst removal) are tested.

As already mentioned, six configurations are used as starting point for the tests on the removal methods, three configurations from the initial tests, combined with two different construction methods for the initial solution. Note that this will result in six different configurations for one removal method that is tested. Therefore, when the average KPIs are reported for a removal method, this is the average taken over 18 configurations: six different configurations for three instances.

8.4.1 Multiple Removal Methods

First of all, the combination of several removal methods is tested, the first three bullets that are described in the introduction. As already mentioned, this can be done with and without a learning layer, or roulette wheel. Recall from Section 6.2, that Ropke and Pisinger (2006b) state that the search benefits from the roulette wheel and the configurations with more removal methods in the roulette wheel find solutions with lower objective function values.

Training Instances B

The average results over all six configurations and three training instances are given in Table 8.8. For comparison between methods, the plan costs are taken as main objective, these are visualized in Figure 8.2, for all cases individually and for the average over all cases. Note that the trips, or routes, and tasks are the summation of all trips and tasks of 18 configuration and case combinations.

Table 8.8: Results Multiple Removal Methods (Average)

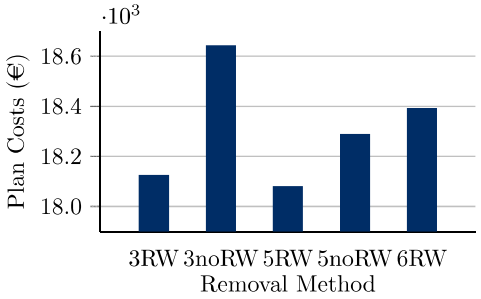
Removal Method	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (h:min)
3RW	735	11,064	18,442	5202	161.91	314,047	3:24
3noRW	741	11,064	18,749	5251	166.14	318,506	2:35
5RW	735	11,064	18,426	5179	161.94	313,070	3:32
5noRW	739	11,064	18,517	5202	162.84	315,881	2:39
6RW	731	11,064	18,558	5200	164.68	313,987	3:07

This table shows the results of tests on five different removal methods, for the training instances B. All methods combine a number of single removal methods, three, five or six, with or without a learning layer (or roulette wheel). Hence, three methods with a learning layer (3RW), three methods without a learning layer (3noRW), five methods with a learning layer (5RW), five methods withouth a learning layer (5noRW) and six methods with a learning layer (6RW). The method that is used in the configuration is stored in the first column. The second and third column are the sum of the trips that are used and the tasks that are planned, respectively. The maximum number of tasks to plan is $(6 \times 794) + (6 \times 794) + (6 \times 256) = 11,064$. The other five columns show results of the other KPIs.

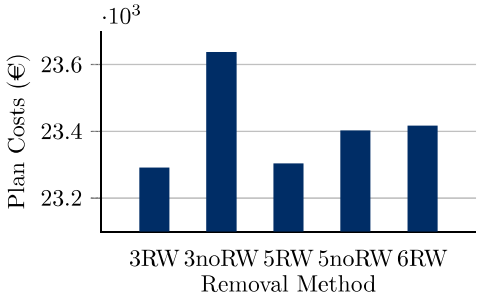
Note that the number of transports, which is the number of tasks divided by two, is the same for all configurations, hence the solutions for the configurations can be compared. Pisinger and Ropke (2007) already concluded that for their test instances, more removal methods combined performs better than having less removal methods combined. This is in line with our results when comparing plan costs, as can be seen in Figure 8.2d. However, the difference between 3RW and 5RW is very small, which can also be seen in Table 8.8. Ropke and Pisinger also mention that the configuration with more removal methods benefits from the roulette wheel that is included. Again, this is in line with the results of our experiments. However, the difference in plan costs is very small for 5RW and 5noRW, less than 0.5%. From Table 8.8 can be seen that all KPIs are around 0.5% better when using a roulette wheel. Only the difference in driving time is slightly bigger, namely 0.9%.

Looking at Figure 8.2d it stands out that the difference between 3RW and 3noRW is quite large. Unfortunately, Ropke and Pisinger do not test a combination of less removal

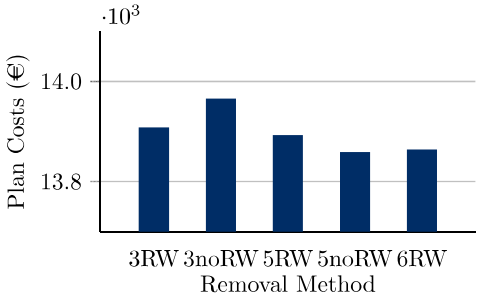
methods without a learning layer, hence we can not compare this with their results. When looking at Table 8.8, one can see that the large difference in plan costs is mainly caused by the difference in hours, which is around 2.5%. The bigger difference between 3RW and 3noRW, with respect to 5RW and 5noRW, might be explained by the fact that with five removal methods more diversification is included. The solutions found without roulette wheel are already closer to the optimal solution than with 3noRW, hence finding improvements with a roulette wheel will be harder.



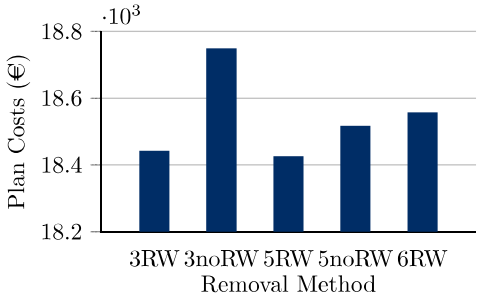
(a) The average plan costs (in euros) per removal method for instance B1.



(b) The average plan costs (in euros) per removal method for instance B3.



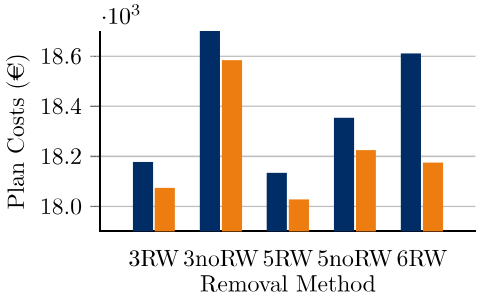
(c) The average plan costs (in euros) per removal method for instance B4.



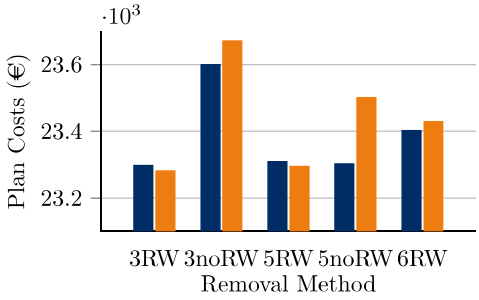
(d) The average plan costs (in euros) per removal method.

Figure 8.2: Four plots of the average plan costs (in euros) per removal method that is tested. The tests are executed on instance B1, B3 and B4. The average over all instances is also plotted. Five removal methods are shown: three removal methods with and without learning layer, 3RW and 3noRW respectively; five removal methods with and without learning layer, 5RW and 5noRW respectively; six removal methods, 6RW. Note that the y axis is in thousands (euros) and the values do not start at zero.

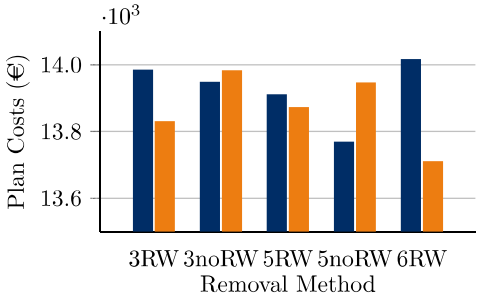
To have more information about the removal methods, graphs are plotted for each training instance in Figure 8.2 a, b and c. As for B1 and B3, they show the same results as mentioned for the average values. Considering Figure 8.2c, the performance of the removal methods is different. It appears to be that for this smaller case with less transports to plan, the influence of the removal method is less than for the larger cases B1 and B3. The differences in plan costs are very small and especially the minor difference between methods with and without learning layer stands out. Ropke and Pisinger (2006b) mention that for smaller cases the difference between using configurations with several removal methods and with only one removal method, is very small. This might also explain the minor differences in what removal strategy is used in our results. For smaller instances, it might be the case that improvements are found no matter what method is used for removing and recreating, as there are less options for moving transports around in the planning. However, it might also be the case that the solutions that are found are already close to the optimal solution and that it is hard for this instance to find more improvements. Unfortunately, we can not compare the gaps between the solutions found with the removal strategies and the optimal solution, as this solution is not known. It would be interesting to tests more on small and



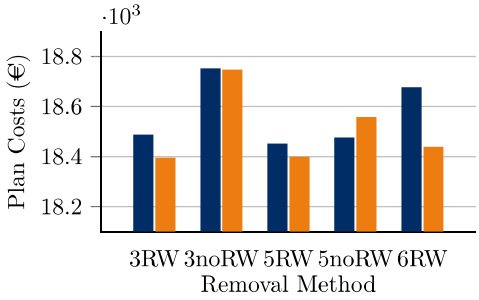
(a) The average plan costs (in euros) per removal method for instance B1.



(b) The average plan costs (in euros) per removal method for instance B3.



(c) The average plan costs (in euros) per removal method for instance B4.



(d) The average plan costs (in euros) per removal method.

Figure 8.3: The average plan costs (in euros) for three instances and the average over all instances. The average is taken over three configurations and is split up for two construction methods for the initial solution: sequential insertion (SI) and parallel insertion (PI). Note that the y axis is in thousands (euros) and the values do not start at zero.

large instances to see the difference in performance.

Furthermore, not much can be concluded on 6RW. It seems that for these instances the difference between using five removal methods and six removal methods is not very big. Although the number of routes that are used with 6RW are less than with 5RW, the KPIs are slightly higher. This indicates that the planning found with 5RW is more efficient.

Note that Table 8.8 and Figures 8.2 show averages that include SI and PI as construction method for the initial solution. When we split this average, it may be concluded that the performance of certain removal methods are depending on the construction method for the initial solution. Consider Figure 8.3. This Figure includes plots for all instances individually, but also the average over all instances are shown in Figure 8.3d. The largest difference can be seen for 5noRW and 6RW, they seem to perform differently in case another construction method for the initial solution is used. With 6RW also all transports on randomly chosen routes are removed with trip removal. It seems to be case specific, whether the way the routs are constructed influence this 6RW removal strategy.

Test Instances C

The tests regarding removal methods are also executed on the training instances C. Looking at Figure 8.4a we see that more removal methods in a roulette wheel provides solutions with slightly lower plan costs than less removal methods in a roulette wheel. Although the differences are very small, the plan costs for 5RW are 0.05% better than for 3RW, this is in line with the findings of Ropke and Pisinger. Furthermore, the gap in plan costs between removal methods with and without a roulette wheel seems remarkable. The differences are again very small: using a roulette wheel with three removal methods decreases the plan

costs with 0.30%, for five removal methods the decrease is 0.43%. Considering the other KPIs in Table 8.9, we see roughly the same decrease in percentages. Even though the gaps are small, this is in line with the results for our training instances B.

Recall from Section 6.2 that Azi et al. (2010) use removals on three levels, including on route level. This is also included in 6RW, where trip removal is included as well. Although the training instances B did not seem to benefit from this extra removal method, 6RW performs slightly better than 5RW for the test instances C. From this we can conclude that the influence of adding an extra removal method in a roulette wheel can be case specific.

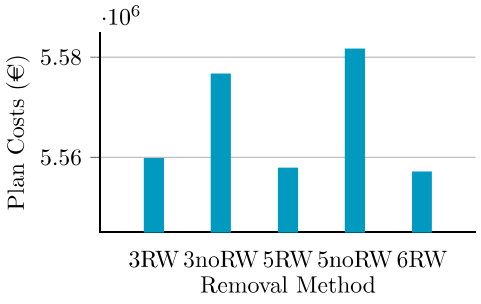
Furthermore, note that the computing time for methods with roulette wheel is longer than for the methods without roulette wheel, more than a factor 2. The only difference is keeping track of the scores and choosing a removal method in the next iteration based on these scores. Further research on this is needed, because it seems odd that the extra computing time is this long.

We also split the results into methods with different constructions for the initial solution in Figure 8.4b. However, the initial solutions does not have a big influence on the performance of these methods.

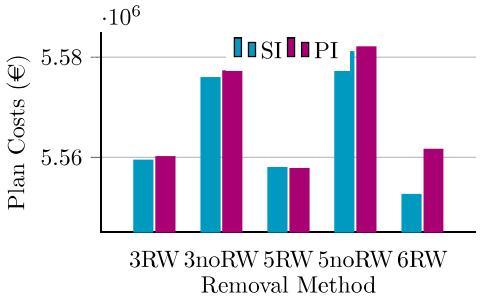
Table 8.9: Results Multiple Removal Methods C (Average)

Removal Method	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
3RW	13,108	49,776	5,559,884	51,353	1966.81	3,692,631	51:49
3noRW	13,150	49,776	5,576,714	51,508	1973.22	3,703,380	22:44
5RW	13,117	49,776	5,557,974	51,335	1965.99	3,691,372	48:51
5noRW	13,151	49,776	5,581,700	51,560	1972.14	3,706,623	22:04
6RW	13,107	49,776	5,557,212	51,323	1968.51	3,690,669	47:35

Each row shows the results of removal strategies with multiple removal methods, with and without roulette wheel, for the test instances C. The average is taken over all 14 instances in combination with six configurations: two different construction methods for the initial solution (SI and PI) and three initial frameworks (93, 109 and 122). Therefore, the total number of tasks that need to be planned is equal to $6 \times 8296 = 49,776$.



(a) The average plan costs (in euros), on the y axis, per removal method for instances C, which are show on the x axis.



(b) The average plan costs (in euros) per removal method, split into the two construction methods for the initial solution, for instances C.

Figure 8.4: The average plan costs (in euros) for the removal strategies with multiple removal methods, with and without learning layer (or roulette wheel), for the instances C. Note that the y axis is in millions and the values do not start at zero.

Conclusion

From the tests that are performed on the training and test instances, we can conclude that the ALNS configuration benefits from a roulette wheel. This is in line with the results of Ropke and Pisinger. Although the difference in KPIs are very small, we can conclude that adding more removal methods in a roulette wheel causes more diversification, which makes it possible to find better local minima during the search. Overall, 5RW can be recommended as removal strategy, where the construction method that is used for the initial solution does not seem to have a big influence on the final solution quality. There are some differences for the two construction methods that are used, but they are small and instance specific.

8.4.2 Single Removal Methods

Besides using several removal methods in one search, with or without roulette wheel, we also test the ALNS with a single removal method. To do so, we still include three different recreate methods, which implies that the metaheuristic still is a ALNS metaheuristic. The experiments with single removal methods are included in this research, to test the added value of using more removal methods during the search. Most of all, it is tested to be able to provide recommendations on what removal method to use in case a configuration is required that is easy to implement and understand. Therefore, we chose the use less iterations for the tests on this component.

Training Instances B

There are articles in the literature that only use one removal method for a predefined number of iterations, such as Shaw (1997). Ropke and Pisinger also test SR, RR and WR as single removal methods. Recall from Section 6.2 that SR provided the best solutions for their test instances, followed by WR and RR. Considering Figure 8.5, we indeed see that SR performs best when looking at the plan costs. Note that, again, these are the average plan costs over three training instances in combination with three different configurations for the rest of the framework. Looking at Table 8.10, the gap between SR and the other methods is big when considering the KPI hour. The difference in hour ranges between 1.4% and 10.4%. Furthermore, the number of routes, or trips, that is needed to plan all transports is around 2% less than the routes that are needed for the other removal methods.

The second and third best removal methods, when considering the average plan costs, are CRR and RR respectively. Removing transports with the use of clusters seems to be beneficial. The difference in plan costs between CRR and SR is small, which may be explained by the configuration of the two methods. For SR, a random seed is chosen to be removed, followed by a predefined number of transports in the neighborhood, where neighborhood is defined with distance. Within CRR, a predefined number of transports is randomly chosen as seed, equal to the number of clusters, followed by a predefined number of transports in the neighborhood, again based on distance. Therefore, CRR can be seen as multiple SRs executed at the same time. It would be interesting to plot the improvements that are found against the number of iterations, to see whether RR performs better when the number of removals is higher.

Furthermore, WR and CWR have the highest plan costs and perform worse than all other removal methods. However, this may be the result of the configuration of these methods. For both methods, the most expensive transports are removed from the solution, where 'most expensive' can be specified within the configuration. However, there is no randomness included. Therefore, it may happen that the same transports remain the 'most expensive' transports in the solution and the same transports will be removed over and over again. Because local search is included, it may happen that at some point this loop of removing the same transports is broken, but local search is only performed when the solution is below a certain threshold. Note that the computing time for WR and CWR is less than for the other methods. This is due to the fact that solutions that are found during the search are stored in the software of ORTEC. Therefore, in case the same transports

are removed, the system recognizes the solution and it will not try to reinsert the removed transports. To obtain more information about the importance of the construction method

Table 8.10: Results Single Removal Methods (Average)

Removal Method	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (h:min)
SR	737	11,064	18,845	5284	167.81	319,696	2:51
RR	756	11,064	19,671	5452	178.59	327,739	2:28
WR	753	11,064	20,183	5542	187.28	331,374	1:44
CRR	750	11,064	19,054	5306	170.19	319,846	2:31
CWR	754	11,064	20,158	5532	186.89	330,500	1:43

The first column stores the single removal method that is used, for the training instances B: related removal (SR), random removal (RR), worst removal (WR), cluster random removal (CRR) and cluster worst removal (CWR). The second and third column show the number of trips that is needed to plan the number of tasks. The maximum number of tasks to plan is $(6\times794)+(6\times794)+(6\times256) = 11,064$. The other five columns show results of the other KPIs that are used to measure the performance of the configurations. Note that for these tests only 4000 iterations of removing and recreating the solution are executed, instead of the 12,000 iterations for other configurations. Therefore, we are not able to comparing these results to other configurations.

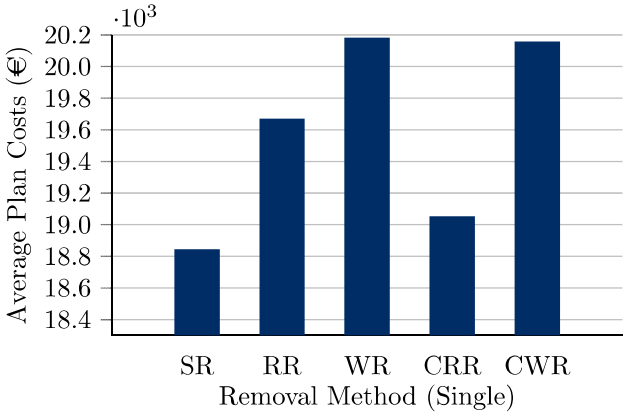
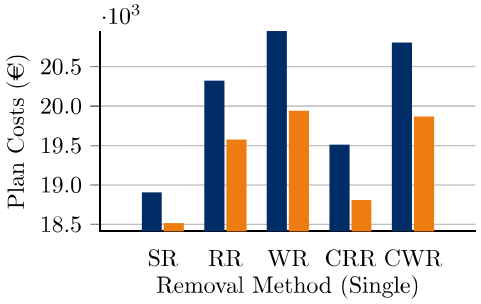
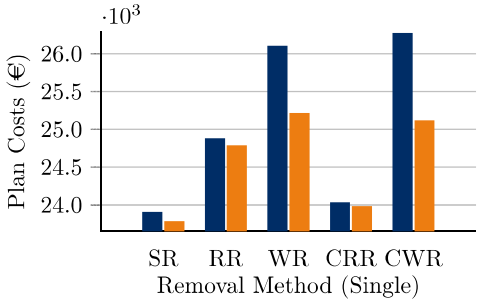


Figure 8.5: The average plan costs (in euros) for configurations that have a fixed removal method: related removal (SR), random removal (RR), worst removal (WR), cluster random removal (CRR) or cluster worst removal (CWR). The average is taken over three instances, two construction methods for the initial solution and three configurations, resulting in nine cases per removal method. The results are from 4000 iterations of removing and recreating the solution. Note that the y axis is in thousands and the values do not start at zero.

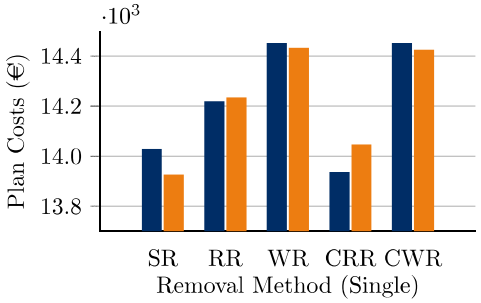
for the initial solution, the average plan costs are split into plan costs for SI and PI in Figure 8.6. Recall that using PI as construction for the initial solution resulted in a better solution than in case SI was used, Table 8.5. From Figure 8.6d we may conclude that a better initial solution will result in a better final solution when only single removal method is used. We can speculate that the use of a single removal method is not strong enough to decrease the gap in performance of the construction methods for the initial solutions. This may be explained by the fact that diversification for neighborhoods is needed and diversification in the number of removals is not enough. However, this may be case specific.



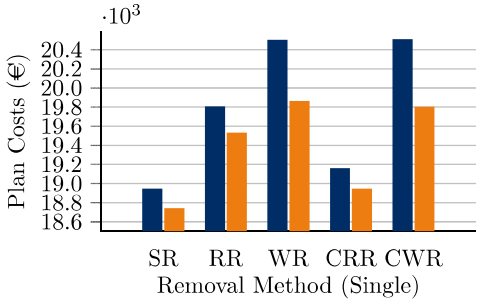
(a) The average plan costs (in euros) for instance B1. Note that the scale on the y axis, steps of 500 euros, differs from the other scales, steps of 200 euros.



(b) The average plan costs (in euros) for instance B3. Note that the scale on the y axis, steps of 500 euros, differs from the other scales, steps of 200 euros.



(c) The average plan costs (in euros) for instance B4.



(d) The average plan costs (in euros) per (single) removal method.

Figure 8.6: The average plan costs (in euros) for three instances and the average over all instances. The average plan costs are taken over three configurations and are split up into two construction methods for the initial solution: sequential insertion (SI) and parallel insertion (PI). The results are from 4000 iterations of removing and recreating the solution. Note that the y axis is in thousands (euros) and the values do not start at zero.

At last, we want to compare the configurations that require multiple removal methods with the ones with a single removal method. To do so, consider Tables 8.8 and 8.10. Recall that the configuration where SR is used as removal method performs best, compared to the other single removal methods. All KPIs of the solution found with SR are higher than the KPIs of all configurations that use multiple removal methods. The increase in KPIs from 5RW to SR, both the best performing methods in case multiple removal methods are used and a single removal method is used, is more than 2%. For the total hours the increase is even higher, around 3.63%. This implies that diversification through the use of structurally different neighborhoods helps the search for finding local minima.

Test Instances C

Considering Table 8.11, we see that using CRR as removal method results in a solution with the lowest plan costs in comparison with the other removal methods. However, the differences in KPIs are very small between CRR and SR. The number of routes that are needed for the planning are even lower for SR. This indicates that the routes that are constructed with SR have more transports on average. It appears that using more routes decreases all KPIs and hence less detours are needed. Although the number of transports that are removed are the same for CRR and SR, the number of transports that are clustered are different. With SR one big cluster is removed, whereas for CRR multiple clusters are removed at once. It might be easier for the software to find improvements when only a few

transports are in a cluster, because there are less options to reinsert the removed transports. However, this can be case specific, as the results are the other way around for instances B: SR performs better than CRR. It might be the case that for the instances C the customer locations are more clustered and hence removing several clusters at once helps the search.

Furthermore, we see in Figure 8.7 that the average plan costs for RR are very close to the average plan costs for SR and CRR. Again, we see that WR and CWR perform by far the worst, when it comes to the average plan costs. The average plan costs increase around 1.4% and 1.5% in comparison with CRR, for WR and CWR respectively. Note that this is due to the configuration of the two methods as is explained above.

We also split the performance of the single removal methods for the two different construction methods SI and PI. Note that SI provided a solution with lower KPIs than PI for the test instances. Apart from RR, we also see this in the single removal methods, although the difference are very small. The increase for using PI as construction method for the initial solution, or decrease for RR, is for all single removal methods less than 0.5%.

At last, we compare the single removal methods with the multiple removal methods that are used during the search. To do so, we compare the two methods from both classes that have the lowest values for the KPIs, hence the best performing methods. Recall that these are 5RW in case multiple removal methods are used during the search, and CRR in case only a single removal method is used. The increase in KPIs from 6RW to CRR is around 0.4%, for all KPIs. Furthermore, the KPIs for CRR are close to the KPIs of the method that performs worst in case multiple removal methods are used during the search, which was 5noRW. Although the differences in KPIs are very small, using multiple removal methods during the search is beneficial for the quality of the solutions.

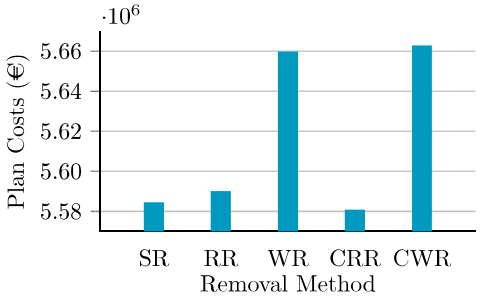
Table 8.11: Results Single Removal Methods C (Average)

Removal Method	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
SR	13,127	49,776	5,584,564	51,581	1976.26	3,707,492	46:38
RR	13,193	49,776	5,590,085	51,635	1976.11	3,712,170	31:56
WR	13,337	49,776	5,659,790	52,276	2002.34	3,753,283	16:46
CRR	13,153	49,776	5,580,903	51,546	1975.02	3,705,721	44:15
CWR	13,336	49,776	5,662,871	52,303	2004.02	3,755,258	19:19

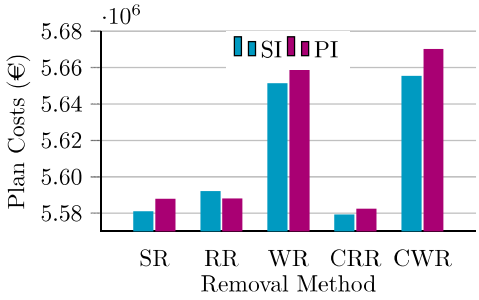
The results for tests regarding single removal methods, for the test instances C: related removal (SR), random removal (RR), worst removal (WR), cluster random removal (CRR) and cluster worst removal (CWR). The maximum number of tasks to plan is $6 \times 8296 = 49,776$. Note that for these tests include 1200 iterations of removing and reinserting transports.

Conclusion

Overall, we can conclude that when using a single removal method, SR and CRR perform best, hence they find solutions with the lowest KPIs. Note that which of the two is best overall, depends on the case that is considered. RR may also be used as single removal method, but the KPIs may be higher, and thus worse, for this method. Note that the performance of CRR as single removal method is not mentioned in the literature. WR and CWR are by far the worst performing single removal methods, but this might be due to the lack of randomness in the configurations of the methods. Furthermore, we speculate that the quality of the construction of the initial solution has a small influence on the quality after using a single removal method, which might be case specific. This is not yet included in the literature that is studied for this research, and needs further research. Nevertheless, Pisinger and Ropke (2007) mention that ALNS performs slightly better than LNS, where the main difference is the use of multiple removal (and recreate) methods during the search with ALNS. This is in line with our results, where the multiple removal methods have lower



(a) The average plan costs (in euros) for configurations that have a fixed removal method during the search. The average is taken over 14 instances and six configurations per instance: two initial solutions and three ALNS frameworks.



(b) The average plan costs (in euros) for configurations with a fixed removal method, split up into the construction methods for the initial solution: sequential insertion (SI) and parallel insertion (PI).

Figure 8.7: The results of the average plan costs (in euros), for instances C, for a fixed removal method followed by removing and reinserting transports. The removal methods are related removal (SR), random removal (RR), worst removal (WR), cluster random removal (CRR) and cluster worst removal (CWR). Note that they y axis are in millions (euros) and the values do not start at zero.

KPIs than the single removal methods. However, the differences are very small for test instances C, smaller than 0.4%, whereas the differences are larger for the training instances B, around 2%. Ropke and Pisinger mention that with an increase in problem size, ALNS performs better than LNS, hence the objective function value is lower. This can be an explanation for the smaller differences for test instancesC, as more smaller instances are included than for the training instances B.

8.4.3 Number of Removals

Not only the kind of removal method is important, but also the number of transports that are removed may influence the performance of the hybrid method. A combination of several values for removing transports will diversify the method, it will help the method to converge to a solution with low KPIs for a minimization problem. However, to understand and test the need for this diversification, it is useful to test with one amount to remove as well. In this section, the number of transports that needs to be removed during each iteration is set to a fixed value. Because these configurations will show differences in the early state of the search already, the total number of iterations is reduced to 250 in the roulette wheel. There is a recursion of two over this roulette wheel, which results in 500 iterations in total.

Training Instances B

Table 8.12 shows the results for seven different percentages of removing transports. Note that, again, these results are averages over three different instances in combination with two different instances (one for SI and one for PI). The number of transports to remove is based on the literature. Ropke and Pisinger (2006b) remove between 10% and 40%, but for larger cases they have a maximum number of transports to remove (Pisinger & Ropke, 2007). Shaw (1997) starts with the removal of 25% of the planned transports. In his method, the number decreases and increases, depending on the performance of the method at that point during the search. For more information, we refer to Appendix B. Furthermore, Schrimpf et al. (2000) also test smaller and larger numbers, ranging from 1% to 50%.

We would expect that the run time increases with the number of removals, so the number of transports that are removed. Removing more transports from the solution, implies having more transports to reinsert, hence a longer run time. This is confirmed by the last column in Table 8.12.

Table 8.12: Results Number of Removals (Average)

Percentage	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (min)
1	249	3688	19,471	5363	177	322,217	6
5	248	3688	18,602	5203	164	314,430	11
10	248	3688	18,667	5213	165	315,128	16
20	249	3688	19,094	5259	172	318,032	29
30	253	3688	18,915	5197	169	313,829	42
40	250	3688	19,214	5283	174	318,467	56
50	251	3688	19,417	5304	177	320,182	73

The percentage of tasks that is removed is given in column one, for the training instances B. The number of iterations is two times 250, so twice a roulette wheel of 250 iterations. Note that due to less iterations we can not compare the results with the other configurations. The second column shows the sum of all trips that are used for the planning. The third column shows all the tasks that are planned, with a maximum number of $(2 \times 794) + (2 \times 794) + (2 \times 256) = 3688$. The last five columns show results for the other KPIs.

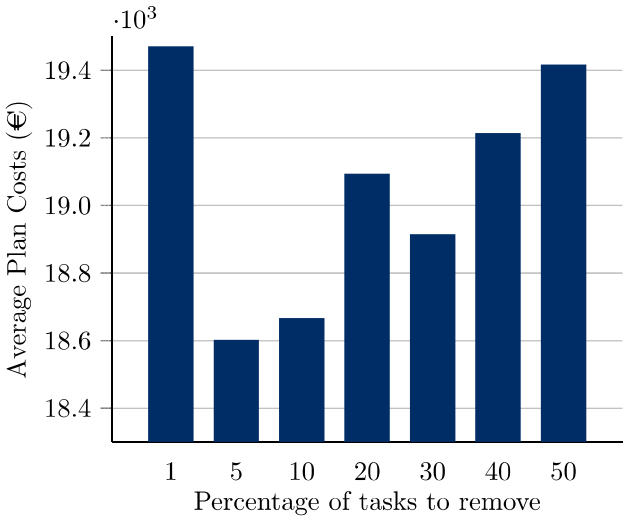
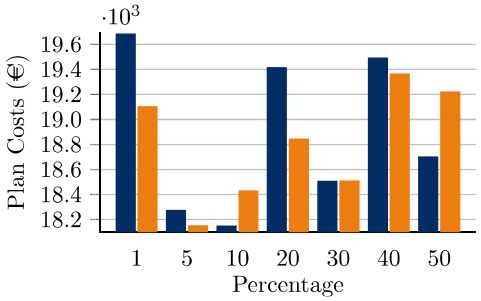


Figure 8.8: The average plan costs (in euros) over three instances and two methods for the construction of the initial solution, per percentage of tasks that are removed during each iteration of removing and recreating the solution. The results are from 500 iterations in total, where a roulette wheel is reset after 250 iterations. Note that the y axis does not start at zero and that the amount of euros is in thousands.

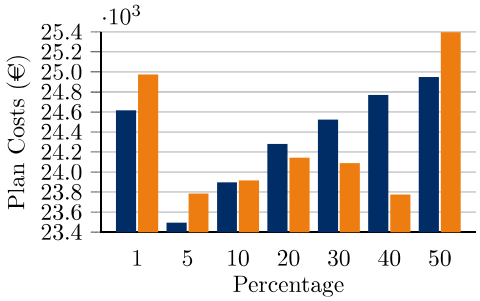
The results regarding the main objective, plan costs, are visualized in Figure 8.8 as well. One can immediately notice that removing 1% of the transports from the solution is not enough. Removing only 1% of the transports will in most cases not lead to an improvement. For example, with RR it is likely that these 1% are scattered all over the network. They will probably be reinserted at the same position they were before they were removed from the solution. Furthermore, it seems to be the case that removing 40% and 50% of the transports is too much. With a high number of transports that are removed, it may be hard to reinsert them all. This may be caused by poor reinsertion methods. Most of the reinsertion methods used for this test, do not include any form of ordering the transports before they are reinserted. PRI has some influence in what transport is reinserted first by

the regret value of not inserting the transport in its best position. The ones with a high regret value will be inserted first. However, because only a little over half of the solution is kept, the regret values may be high for many transports. Therefore, a more intelligent method for reinsertion is needed.

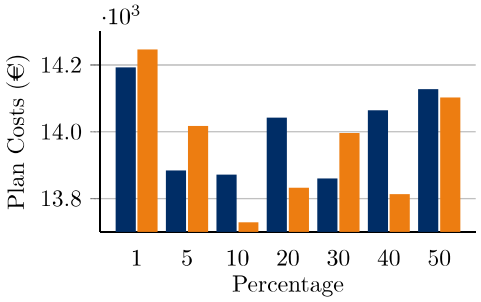
Moreover, 5% and 10% are the numbers that result in the best solutions for these instances. However, looking at Figure 8.9 this seems to be case-specific. Overall, 5% and 10% still belong to the best solutions, regarding the average plan costs, but the difference with higher percentages is not always that big. For example, in Figures 8.9b and 8.9c, 40% seems to be a good number of transports to remove as well, in case PI is used as the method for constructing the initial solution. These figures also show that the construction method for the initial solution has influence on the total performance as well. Especially for B3, the pattern of performance when the percentage of removals is increased is different for SI and PI as construction methods for the initial solution. For SI it seems that a higher percentage to remove increases the KPIs, whereas for PI there is a dip for 30% and 40%.



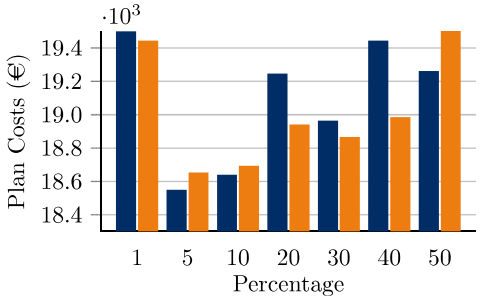
(a) The average plan costs (in euros) per percentage of tasks that need to be removed, for instance B1.



(b) The average plan costs (in euros) per percentage of tasks that need to be removed, for instance B3.



(c) The average plan costs (in euros) per percentage of tasks that need to be removed, for instance B4.



(d) The average plan costs (in euros) per percentage of tasks that need to be removed.

Figure 8.9: The average plan costs, in euros, per percentage of tasks that need to be removed, split up into two configurations with different construction methods for the initial solution: sequential insertion (SI) and parallel insertion (PI). The results are from 500 iterations of removing and recreating the solution, where the number of tasks that is removed is equal to the percentage that is given on the x axis. Note that the values on the y axis are in thousands and that the values do not start at zero.

Test Instances C

We also executed tests with a single percentage of transports that needs to be removed from the solution during the search on the test instances C. From Figure 8.10a it is clear that removing 1% of the transports in the network is not enough for the search to find

improvements. When only a few transports are removed, it is likely that they will be reinserted in the same place from before they where removed. Interchanging the removed transports is in most cases not feasible and if they can be interchanged, it it likely that the objective function value will not decrease much as only small moves are possible.

Increasing the percentage of removals to 5% results in a decrease of the average plan costs of around 1.07%, which can be seen in Table 8.13. All other KPIs also decrease with a little more over 1%. However, the computing time increases with a factor around 3.2. This is as expected, because the higher the percentage of transports that are removed, the higher will be the number of insertion possibilities per transports. That indicates that the computing time will increase with the number of transports that are removed. The average plan costs decrease slightly between removing 5% and 10% of the transports. From there on, an increase in the percentage of transports that is removed implies an increase of the average plan costs as well. This pattern was already shown for the training instances B, but is more clear for the test instances C as we have more instances in this case and extremes will be evened out. Nevertheless, the computing time for removing 10%, 20% and 30% of the transports is roughly the same and increases only for 40% and 50%.

Furthermore, to see the influence of the construction method for the initial solution, we included Figure 8.10b. From this figure, we see that the construction method that provided a solution with lower plan costs, SI, also performs best when we test on the percentage of transports that need to be removed. Because we do not vary the number of transports that is removed during the search in these tests, we take away a part of the diversification that is included in ALNS. We see in Figure 8.10b that this slightly influences the ability of ALNS to find improvements for initial solutions that have slightly higher plan costs. Note that the increase in plan costs between using SI and PI as construction method for the initial solution is less than 1%.

Table 8.13: Results Number of Removals C (Average)

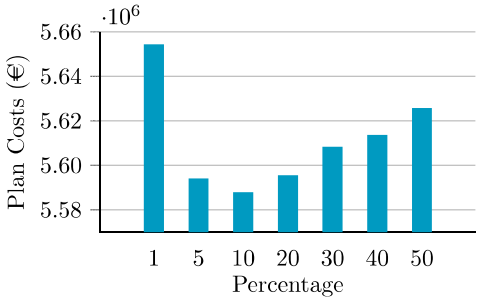
Percentage	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
1	4441	16,592	5,654,368	52,221	2002.79	3,751,480	02:53
5	4396	16,592	5,594,088	51,664	1981.61	3,714,134	09:14
10	4379	16,592	5,587,938	51,607	1979.66	3,709,634	11:05
20	4390	16,592	5,595,558	51,684	1979.10	3,714,226	11:00
30	4394	16,592	5,608,332	51,803	1983.48	3,722,696	11:08
40	4403	16,592	5,613,670	51,849	1986.65	3,725,276	13:57
50	4419	16,592	5,625,734	51,950	1995.94	3,732,174	16:17

The results of using a fixed number of transports that are removed during each iteration of the search, for the test instances C. A roulette wheel of 250 iterations is used twice, hence a total of 500 ruin and recreate iterations are executed. The percentage of transports that is removed from the solution during each iteration is given in the first column. The results are averages over 14 instances in combination with two initial solutions, which results in the sum of total transports of $2 \times 8296 = 16,592$.

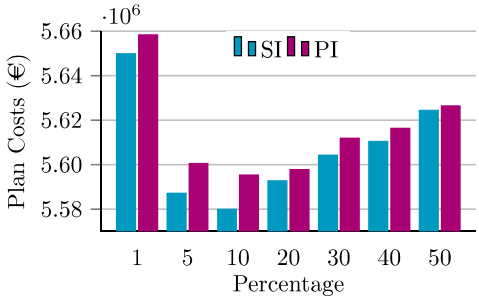
Conclusion

We can conclude that having only 1% of the transports removed is not enough for the hybrid method to find improvements. The same holds for 50%, which is too much to remove from the solution. The higher percentages of transports that are removed may perform better with a reinsertion method that takes into account the characteristics of the transports. For example, reinserting them with the use of the reinsertion methods that are used for SI or PI may lead to better solutions.

From 5% of the transports that are removed till 50%, the plan costs generally increase. The decrease in plan costs from removing 1% to 5% is in line with the findings of Schrimpf



(a) The average plan costs (in euros), on the y axis, per percentage of transports that need to be removed. The x axis show the percentages.



(b) Results of the average plan costs (in euros) per percentage of transports that need to be removed, split up for two construction methods for the initial solution.

Figure 8.10: The average plan costs (in euros), for instances C, for a fixed percentage of transports that are removed in each iteration during the search. Note that the values on the y axis are in millions (euros) and that the values do not start at zero.

et al. (2000). In case only one number is used for removing transports, we recommend to use a percentage between 5% and 10%, as these two numbers seem to produce solutions of good quality for all cases. Combining several percentages, for example with an interval, we recommend a minimum of 5% and a maximum of 20%.

8.5 Recreate Methods

The third component of the hybrid method is the recreate method. The configurations that are tested have many iterations and in each iteration the recreate method is executed. Therefore, we choose to only test with simple recreate methods: PCI, PRI and CI. As already explained in Section 8.4.3, it may be beneficial to have a more sophisticated recreate method when the number of transports that is removed is large. Within the configurations that are tested for the recreate method, the maximum percentage of transports that are removed is around 20%. Therefore, simple recreate methods should be sufficient.

For the recreate method configurations, we test with two construction methods for the initial solution (SI and PI) and one removal strategy: 5RW. This results in 18 different configurations per recreate method: two methods for constructing the initial solution, three configurations (93, 109 and 122) for three different training instances. We also executed tests with removal strategy SR for the instances B. The results were not different from the ones with strategy 5RW and therefore the figures and table are included in Appendix C.

Training Instances B

Table 8.14 shows the results of the tests performed on the recreate methods PCI, PRI and CI, with 5RW as removal strategy. Considering the plan costs, that are also visualized in Figure 8.11, there are only minor differences. It stands out that the plan costs for PCI and CI are on average the same. However, this is misleading. Consider Figure 8.12 and compare PCI and CI for each instance. The two recreate methods do perform differently and it is case specific which performs best. Although there are some differences, also for SI and PI as construction method for the initial solution, these differences are very small.

Returning to Table 8.14 and Figure 8.11, we see that PRI performs slightly better when it comes to the plan costs. The distance that is driven in the network is slightly more than with PCI and CI, but the hours are slightly less. Furthermore, the number of trips, or routes, that are used are less for PRI as well. Therefore, we may conclude that PRI performs best as recreate method when considering the KPIs. Unfortunately, this method comes with the price of a run time that is longer than the run time for the other two recreate

Table 8.14: Results Recreate Method on 5RW (Average)

Recreate Method	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (h:min)
PCI	742	11,064	18,533	5188	163.19	313,074	3:05
PRI	740	11,064	18,482	5194	162.30	313,301	3:39
CI	741	11,064	18,533	5187	163.34	313,082	2:07

This table shows the results of having a fixed method for the recreate method, where five methods with a learning layer are used to remove transports, for the training instances B. The different recreate methods are: parallel cheapest insertion (PCI), parallel regret insertion (PRI) and cheapest insertion (CI). The second and third column show the sum of the total number of trips that are used and tasks that are planned, respectively. The maximum number of tasks to plan is $(6\times794)+(6\times794)+(6\times256) = 11,064$. The last five columns contain information about the other KPIs that are used to indicate the performance of the configurations.

methods. These are the results of PRI being a more complicated recreate method, because it takes into account the additional costs of inserting a transport in its second best place. These results are in line with the results of Ropke and Pisinger (2006a), who state that simple recreate methods perform worse.

Recall the KPIs from Table 8.9 for 5RW. This strategy includes three recreate methods in the roulette wheel as well. The KPIs of the solution when only PRI is used as recreate method are only slightly higher. Hence, the diversification of the search when it comes to the recreate method benefits the search, although the differences between using multiple recreate methods or just one are very small.

Unfortunately, the literature does not state the influence of the construction method for the initial solution on the performance of the recreate method, as far as the authors knowledge goes. However, considering Figure 8.12d, we speculate that there is some influence noticeable. Figure 8.12d shows that SI contributes to lower plan costs than PI.

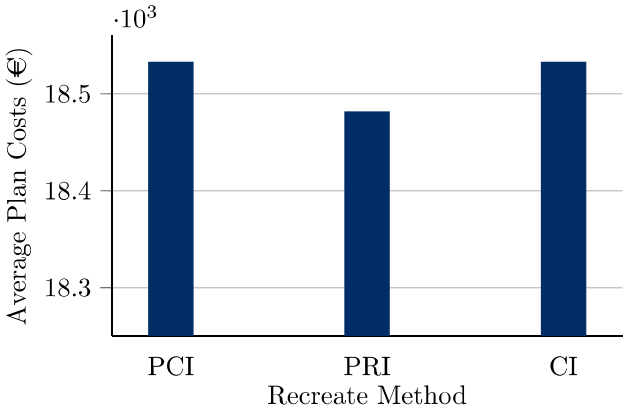
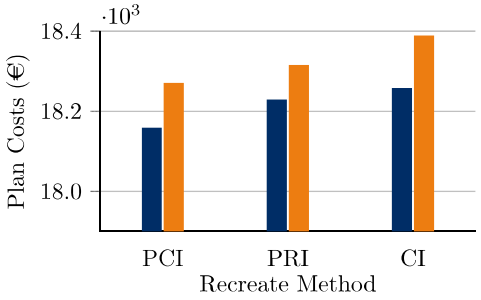
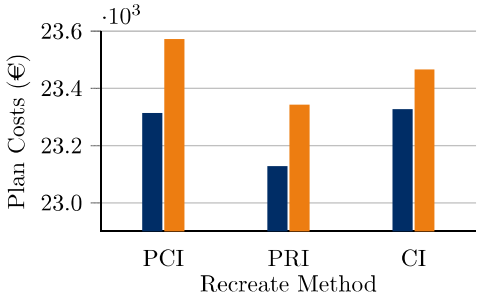


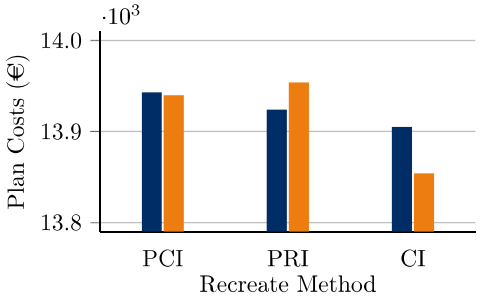
Figure 8.11: The average plan costs, in euros, for three different fixed recreate methods: parallel cheapest insertion (PCI), parallel regret insertion (PRI) and cheapest insertion (CI). The average is taken over three instances, two methods for constructing the initial solution and three configurations, resulting in 18 different cases per recreate method. For the removal component, five removal methods are used with a roulette wheel (5RW). Note that the values on the y axis do not start at zero and the scale is in thousands.



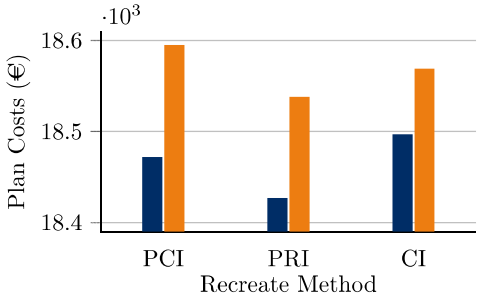
(a) The average plan costs (in euros) per recreate method for the instance B1.



(b) The average plan costs (in euros) per recreate method for the instance B3.



(c) The average plan costs (in euros) per recreate method for the instance B4. Note that the scale on the y axis, steps of 100 euros, differs from the other scales, steps of 200 euros.



(d) The average plan costs (in euros) per recreate method. Note that the scale on the y axis, steps of 100 euros, differs from the other scales, steps of 200 euros.

Figure 8.12: The average plan costs (in euros) for three instances and the average over all instances per recreate methods, split up into two configurations for the construction of the initial solution: sequential insertion (SI) and parallel insertion (PI). For the removal component, five removal methods are used with a roulette wheel. Note that the y axis are in thousands and the values do not start at zero.

Test Instances C

The test on the recreate method are also executed on the test instances C. Considering Table 8.15, we see that the plan costs vary very little when it comes to the recreate methods. The increase in KPIs from PCI to PRI is smaller than 0.1%. From this we can conclude that the influence of the single recreate method that is used is not big. Comparing the KPIs distance and hours, we see that PCI is slightly better: the distance is less than with the other recreate methods. The hours for PCI and CI are almost the same, but because of the difference in distance, we see that the wait time when using PCI is less. Hence, as we would expect, parallel insertion is a better method for insertion, comparing PCI and CI.

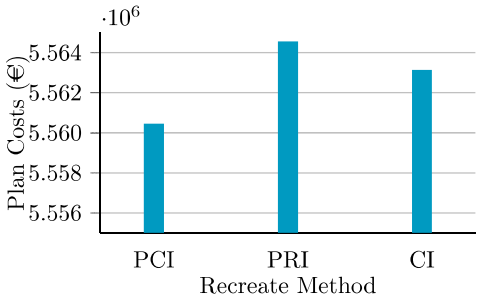
Furthermore, when we split the results for the two construction methods that are used for the initial solution, only minor differences can be seen. Looking at Figure 8.13b, we needed up to thousands of the plan costs, that are in millions in this graph, to see the minor differences.

Recall the values for the 5RW removal strategy, from Table 8.9, where the three recreate methods are chosen in each iteration based on the scores in a roulette wheel. Although the differences are minor, we can conclude that the use of three recreate methods in the roulette wheel improves the solution. The improvement is only around 0.1%, but the diversification helps the search a bit.

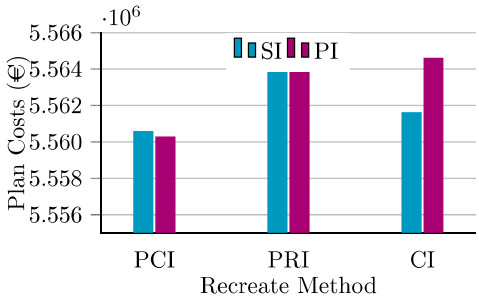
Table 8.15: Results Recreate Methods C on 5RW (Average)

Recreate Method	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hours (h)	Driving Time (sec)	Run Time (min:sec)
PCI	13,103	49,776	5,560,449	51,357	1967.81	3,692,894	49:25
PRI	13,119	49,776	5,564,562	51,393	1970.14	3,695,635	45:51
CI	13,115	49,776	5,563,134	51,385	1967.21	3,694,816	44:24

The results of a fixed recreate method, where five removal methods with a learning layer, or roulette wheel, are used as removal strategy, for the test instances C. Each row represents a fixed recreate method: parallel cheapest insertion (PCI), parallel regret insertion (PRI) and cheapest insertion (CI). The total number of tasks that need to be planned is $6 \times 8296 = 49,776$, as the average is taken over all 14 instances in combination with six ALNS frameworks.



(a) The results of the average plan costs (in euros) on the y axis per fixed recreate method during the search, on the x axis.



(b) The average plan costs (in euros) for a fixed recreate method during the search, split up for the two construction methods for the initial solution.

Figure 8.13: The average plan costs (in euros), where the recreate method is fixed during the search. Three different recreate methods are used: parallel cheapest insertion (PCI), parallel regret insertion (PRI) and parallel cheapest insertion (CI). Note that the y axis is in millions (euros) and the values do not start at zero. Also note that we needed three digits to show the different values on the y axis. This indicates that the differences between the methods are very small.

Conclusion

Overall, we can conclude that including three recreate methods in a roulette wheel helps the search in finding improvements. Although the differences are very small, when only one recreate method is used the KPIs are slightly higher. In case only one recreate method can be used, we advise to use either PCI or PRI. CI seems to be a reinsertion method that contains to much randomness and less improvements are found. Whether PCI or PRI needs to be used as recreate method is case specific. Furthermore, we speculate that when a large number of transports is removed from the solution, more sophisticated reinsertion methods might be useful to find more improvements.

8.6 ORTEC Adaptive Large Neighborhood Search

From the tests that are reported on in the previous sections, we can construct a hybrid method where the components are configured with the methods that resulted in solutions with the best KPIs. As mentioned in Chapter 7, we will provide two configurations. The first one will be the configuration with the best objective function value, based on the KPIs, which will be our hybrid ALNS. The second configuration is a more simple heuristic, where only one removal and one recreate method are used in the search. Therefore, this more simple configuration can be seen as a variant of LNS.

Based on the conclusions per component that are given in previous sections, the following configurations are chosen as the configurations with the best performing components.

- Five removal methods with a roulette wheel (5RW) in combination with three recreate methods in the same roulette wheel (3RW), for a total of 1200 iterations.
- Related removal (SR) in combination with parallel cheapest insertion (PCI), for a total of 500 iterations.

Recall the different frameworks that came out of the initial testing, 93, 109 and 122. In many cases 122, recall that this configuration uses an alternative local search, provided the solution with the lowest KPIs. However, the computing time was longer than for the other two configurations. Therefore, we do not include this configuration in further testing. So far, we only tested with two methods for the number of transports that are removed during the search: 30, 45, 60 transports and 130, 110, 90, 70, 50, 30 transports. Now that the final configurations are known, additional tests regarding the number of transports can be executed. To do so, we will first test on the hybrid method with percentages of transports that need to be removed and we will test both a decreasing and increasing order:

- 5%, 12% and 20% (and reversed),
- 5%, 8%, 12%, 15%, 18%, 22% (and reversed).

These percentages are based on the results reported on in Section 8.4.3. Consider Table 8.16, where the average results over both percentage methods, that are described above, are given. Here, the results are shown for increasing/decreasing the number of transports that are removed during the search. Note that these configurations have a total of 1200 iterations, which consist of two times 600 iterations. For example, 200 iterations are executed for all percentage levels 5%, 12% and 20%, which results in a total of 600 iterations, followed by another round of 5%, 12% and 20% iterations, which results in a total of 1200 iterations.

Table 8.16: Results Number of Transports to Remove (Average)

Case	Trips	Tasks	Plan Costs (€)	Distance (km/ miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
Increasing B	490	7376	18,333	5161	160.42	311,944	30:57
Decreasing B	497	7376	18,548	5164	163.61	311,560	30:31
Increasing C	8750	33,184	5,563,516	51,388	1967.22	3,695,042	23:34
Decreasing C	8747	33,184	5,568,163	51,428	1970.75	3,697,908	25:07

The average results for the hybrid method with five removal methods and three recreate methods, with roulette wheel, when the number of transports that is removed is increased or decreased during the search. The results are given for both the training instances B and test instances C. Note that the average is taken over four configurations: two construction methods for the initial solution and two percentage strategies. The total number of transports to plan is thus $4 \times 1844 = 7376$ for the case B, and $4 \times 8296 = 33,184$ for W.

Looking at the results in Table 8.16, we see that for both the training instances B and the test instances C the KPIs increase when the number of transports that are removed are decreased during the search, compared to when the number of transports that are removed are increased during the search. This implies that increasing the number of transports that are removed during the search results in solutions of better quality. The differences are bigger for the training instances B, than for the test instances C. The results can be explained by the fact that at the beginning of the search improvements can be found rather easy, hence not many transports need to be removed and reinserted. However, further in the search, larger neighborhoods need to be explored as well, to escape local minima. This is in line with Shaw (1997). However, Ropke and Pisinger (2006a) mention that at the

Table 8.17: Results Percentage to Remove (Average)

Percentage	Case	Trips	Tasks	Plan Costs (€)	Distance (km/ miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
5, 12, 20	B	244	3688	18,339	5171	160.63	312,354	25:17
5,$\cdots$, 22	B	246	3688	18,327	5151	160.21	311,534	36:38
20, 12, 5	B	248	3688	18,664	5177	165.90	312,305	24:59
22,$\cdots$, 5	B	249	3688	18,432	5151	161.32	310,816	36:03
5, 12, 20	C	4377	16,592	5,566,631	51,412	1970.85	3,696,084	21:05
5,$\cdots$, 22	C	4373	16,592	5,560,402	51,364	1963.60	3,693,999	26:04
20, 12, 5	C	4376	16,592	5,567,435	51,419	1971.64	3,697,430	22:50
22,$\cdots$, 5	C	4371	16,592	5,568,891	51,437	1969.86	3,698,386	27:24

The average results for the hybrid method for percentage strategies that can be used. There are four options for the percentage of transports that are removed during the search: 5%, 12% and 20%, and reversed, or 5%, 8%, 12%, 15%, 18%, 22%, and reversed. The total number of transports to plan is $2 \times 1844 = 3688$ for the case B, and $2 \times 8296 = 16,592$ for W, as the average is taking over two configurations with different initial solutions.

end of the search not many improvements are found, and thus the number of transports that are removed need to be of decreasing order. Therefore, in the middle of the search, we start with the smallest number of transports that need to be removed and increase the number again during the next iterations. With this method, we combine smaller and larger neighborhoods that are explored during the search.

Furthermore, considering Table 8.17, we split up the increasing and decreasing order in the actual percentages that are used. In all cases, except for decreasing order for instances C, it seems to be beneficial to enlarge or reduce the neighborhoods sooner in the search. With less percentage variation, more iterations are performed with the same number of transports that are removed, with respect to more variation in the percentages. Diversification with respect to the size of the neighborhoods improves the search for near-optimal solution, which is in line with the literature. However, the computing time seems to increase when more variation is used in the percentages, especially for the instances B. The only difference is the number of roulette wheels that are used, as we configured for each percentage a new roulette wheel. The start of such roulette wheel seems to take up a lot of time. However, with the instances C the difference in computing time is less. Nevertheless, this needs some further investigation.

Moreover, we split up the results for the two initial solutions: sequential insertion and parallel insertion. For these additional tests and instances, the differences in KPIs are minor. What construction method for the initial solution found slightly better solutions with removing and reinserting transports, depends on the cases. The table with the results can be found in Appendix D.

We did not only execute additional tests on the hybrid method, but also on the more simple variant with large neighborhood search. From the tests that are reported on in Section 8.4.3, we found that the configuration with removing 10% of the transports in each iteration, and reinserting them again, was able to find the lowest KPIs for the instances. Therefore, we use 10% as a fixed percentage of transports that need to be removed during the search for the more simple heuristic. However, to see the influence of searching through larger neighborhoods and adding diversification in the size of the neighborhoods, we also test with altering between 10% and 20%. For each percentage we include 125 iterations and after 20% we reduce the neighborhood again to 10%. Therefore, we have a total of $4 \times 125 = 500$ iterations. We did not include tests with a decreasing order of number of transports that need to be removed, due to the results of the tests that were executed with

the hybrid method. The results are shown in Table 8.18. We would expect that adding

Table 8.18: Results Percentage to Remove Simple LNS (Average)

Percentage	Case	Trips	Tasks	Plan Costs (€)	Distance (km/ miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
10	B	242	3688	18,810	5240	169.24	316,176	16:51
10, 20	B	245	3688	19,027	5275	171.91	318,547	23:55
10	C	4381	16592	5,589,004	51628	1974.37	3,710,076	11:50
10, 20	C	4381	16592	5,585,009	51584	1976.36	3,708,395	14:16

The results of different percentages of transports that are removed during the search, for a simple LNS with related removal and parallel cheapest insertion. The rows show averages over all instances per case and over two construction methods for the initial solution. Hence, the total number of transports that needs to be planned is $2 \times 1844 = 3688$ for the instances B and $2 \times 8296 = 16,592$ for the instances C.

another level of percentage would decrease the KPIs, as larger neighborhoods help escape local minima. Indeed this is the case for instances C, although the differences are very small. However, this is not the case for instances B. This might be explained by the number of iterations that is less for this simple heuristic. We can speculate that we move to a larger neighborhood to soon in the search, and more improvements can be found with the smaller neighborhood. However, this seems to be case specific. Because this configuration is used as a simple variant of LNS, we advise to use 10% as fixed number of transports that needs to be removed.

Moreover, we split the results up for the two insertion methods. However, just as with the hybrid method, the results are very close. What construction method for the initial solution contributes to a better solution after removing and reinserting transports is case specific. Further research needs to be done to see whether the cases have characteristics that influence the behavior of the search. The table with the results are included in Appendix D.

Furthermore, as we would expect, the results with this simple heuristic are worse than with the hybrid methods from the previous tables in this section. Diversification in both the neighborhood size and methods that are used for the components is needed to find improvements.

To conclude this chapter, two configurations are advised to use, one hybrid method that we call ORTEC Adaptive Large Neighborhood Search, and one simple variant of LNS, called ORTEC Large Neighborhood Search.

- Five removal methods with a roulette wheel (5RW: related removal, random removal, worst removal, cluster random removal, cluster worst removal) in combination with three recreate methods in the same roulette wheel (3RW: parallel cheapest insertion, parallel regret insertion, cheapest insertion), for a total of 1200 iterations, where the number of transports that is removed increases as follows: 5%, 8%, 12%, 15%, 18%, 22%.
- Related removal (SR) in combination with parallel cheapest insertion (PCI), for a total of 500 iterations, where the number of transports that is removed is 10%.

An overview of the KPIs for the construction method for the initial solution, ORTEC ALNS and ORTEC LNS is given in Table 8.19 for instances B and in Table 8.22 for instances C. Note that the KPIs for the original customer configuration is also included for the instances B. The improvements, in percentages, that are found with ORTEC ALNS and ORTEC LNS, with respect to the original customer configuration for instances B is given in Table 8.20. To see the improvements compared to the initial solution that is constructed, an overview

is given in Table 8.21. Compared to the original customer configuration, we can find great improvements with ORTEC ALNS. The computing time is a little over 36 minutes, which is acceptable for the customer. With a more simple improvement heuristic, ORTEC LNS, the improvements regarding the distance and driving time are still big. However, the KPI hour did increase. Note that we took the original customer configuration, but adjusted the cost set in the input data. When we compare it with the original data from Table 8.1, the hours did decrease with ORTEC LNS.

Table 8.19: Results comparison for instances B

Configuration	Trips	Tasks	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
Original	121	1840	18,914	5462	167.45	327,452	02:35
Initial Solution	126.5	1844	20,766	5755	193.65	342,813	03:32
ORTEC ALNS	123	1844	18,327	5151	160.21	311,534	36:38
ORTEC LNS	123	1844	18,810	5240	169.24	316,176	16:51

The KPIs for the original customer configuration are provided in the first row. The second row shows the KPIs for the initial solution without large neighborhood search. Note that the average is taken over SI and PI. The third and fourth row show results for ORTEC ALNS and ORTEC LNS, respectively, as averages for two construction methods for the initial solution: SI and PI.

Table 8.20: Results for instances B w.r.t. Original Configuration (in percentage)

Configuration	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)
ORTEC ALNS	-3.10%	-5.69%	-4.32%	-4.86%
ORTEC LNS	-0.55%	-4.06%	1.07%	-3.44%

The improvements, in percentages, for instances B when ORTEC ALNS and ORTEC LNS are used, compared to the original customer configuration.

Consider Tables 8.22 and 8.23. The improvements that were found with ORTEC ALNS and ORTEC LNS, compared to the initial solution, are between 1.36% and 2.00% for all KPIs. For these instances, the difference between a simple improvement heuristic and more sophisticated heuristic is smaller. However, the improvement is still 0.5%, which implies saving a lot of money for big companies.

Table 8.21: Results for instances B w.r.t. Initial Solution (in percentage)

Configuration	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)
ORTEC ALNS	-11.74%	-10.49%	-17.20%	-9.12%
ORTEC LNS	-9.42%	-8.94%	-12.54%	-7.77%

The improvements, in percentages, for instances B when ORTEC ALNS and ORTEC LNS are used, compared to the initial solution without removing and reinserting transports.

Table 8.22: Results comparison for instances C

Configuration	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
Initial Solution	2228.5	8296	5,673,626	52,398	2010.05	3,761,414	01:23
ORTEC ALNS	2186.5	8296	5,560,402	51,364	1963.60	3,693,999	26:04
ORTEC LNS	2190.5	8296	5,589,004	51,628	1974.37	3,710,076	11:50

The first row shows the KPIs for the initial solution without large neighborhood search. Note that the average is taken over SI and PI. The second and third row show results for ORTEC ALNS and ORTEC LNS, respectively, as averages for two construction methods for the initial solution: SI and PI.

Table 8.23: Results for instances C w.r.t. Initial Solution (in percentage)

Configuration	Plan Costs (€)	Distance (miles)	Hour (h)	Driving Time (sec)
ORTEC ALNS	-2.00%	-1.97%	-2.31%	-1.79%
ORTEC LNS	-1.49%	-1.47%	-1.77%	-1.36%

The improvements, in percentages, for instances C when ORTEC ALNS and ORTEC LNS are used, compared to the initial solution without removing and reinserting transports.

Chapter 9

Conclusion and Recommendations

This chapter includes a recap of the results that are found in our research on how the components of the Adaptive Large Neighborhood Search need to be configured for real-world cases of a VRP. We will answer the subquestions in order to answer the main question of this thesis report. Recommendations for further research and for ORTEC are provided as well.

We have seen many variants of the VRP in Chapter 2 that are all related to the VRP that we reported on in this thesis. Usually, these VRPs are solved with mathematical models, or these models provide bounds for the optimal solution, as explained in Chapter 3. For the software of ORTEC that is used for this research, such a model was not required and therefore we did not provide it. To be able to find near-optimal solutions for the real-world cases, minima-escaping heuristics of the local search class were used. We have seen examples in Chapter 4 such as Simulated Annealing, Tabu Search, Iterated Local Search and Variable Neighborhood Search. These heuristics are all used to search the solution space in a clever way and to be able to escape local minima. This is done by variations in the solutions that are accepted and all kinds of perturbations in the solution at some point during the search. We have seen three examples in Section 4.4, for rich VRPs, for which transports are removed from an initial solution and reinserted to find improvements. These methods use large neighborhoods, in case a large part of the solution is altered. The first one was Large Neighborhood Search by Shaw (1997), where one removal, related removal, and one recreate method were used to find improvements. Schrimpf et al. (2000) presented a similar heuristic, Ruin and Recreate, where the removal and recreate method are different. The third one is based on Large Neighborhood Search, reported on by Ropke and Pisinger (2006a, 2006b, 2007), called Adaptive Large Neighborhood Search. As multiple removal and recreate methods are used, with the help of a learning layer, they state that their heuristic is able to adapt to the problem that is looked into. Therefore, this thesis reported on how to configure such an Adaptive Large Neighborhood Search, with one of the software for optimization of ORTEC, for real-world cases with a limited computing time.

Several components of the Adaptive Large Neighborhood Search (ALNS) were tested and reported on, including the construction method for the initial solution, removal methods for removing transports from the solution, recreate methods to reinsert them, and how to incorporate local search to intensify the search. We will answer the subquestions in the next sections.

Initial Solution

Which construction method is needed to build an initial feasible solution, such that Adaptive Large Neighborhood Search performs the best?

From the literature study in Section 6.1 we have seen that an initial solution with a low objective function value should converge faster to a near-optimal solution (Schrimpf et al., 2000). However, the results of the ALNS regarding the choice for the construction method for the initial solution are not mentioned. We have seen that most studies use a simple insertion heuristic, such as cheapest insertion, or they assume that an initial solution is provided as input for the heuristic. In our study, the construction of the initial solution is a component in the heuristic itself.

We saw in Section 8.3 that the simple insertion methods were not able to plan all the transports for our instances. Therefore, we tested with an ordering of transports that needed to be inserted with sequential and parallel insertion. Our results showed that using parallel insertion as construction method for the initial solution, with ALNS, resulted in average solutions with the lowest KPIs. However, parallel insertion did not result in the best initial solution without ALNS for all instances. Hence, we cannot conclude whether an initial solution with low KPIs also contributes to a solution with low KPIs in case the solution is ruined and recreated. The influence of the quality of the initial solution on the quality of the solutions found with ALNS seems to be case specific.

Removal Methods

Which removal methods and corresponding settings are needed, such that Adaptive Large Neighborhood Search performs the best?

The second component of ALNS that was tested on, was which removal method needs to be chosen. We concluded from the tests, that were reported on in Section 8.4, that in case a single removal method is used, related removal and cluster random removal provided solutions with the lowest KPIs. That related removal performs best is in line with the literature that was provided in Section 6.2, but cluster random removal is not reported on. Which one of the two methods provided solutions with the lowest KPIs was case specific. Furthermore, we speculated that the construction method for the initial solution has some influence on ALNS: initial solutions with low KPIs also resulted in solutions with low KPIs after removing and reinserting transports.

In case multiple removal methods can be used during the search, we found that a combination of five removal methods in a roulette wheel resulted in solutions with the lowest KPIs. However, the differences with more and less removal methods in a roulette wheel were small. We concluded that the heuristic benefits from the roulette wheel, which is in line with the literature. Furthermore, we found that the quality of the initial solution did not seem to have an effect on the performance of the roulette wheel. Only minor differences could be found, which were both case and instance specific.

Regarding the number of transports that need to be removed, we found that using 5% or 10% as fixed number resulted in solutions with the lowest KPIs. On average, the KPIs increased in case the percentage of removals increased from 10% to 50%. In case multiple numbers of transports that need to be removed is used, we recommend numbers between 5% and 20% to increase the size of the neighborhoods that are searched through. These numbers are slightly less than the numbers that were found in the literature, they got up to 40%, which might be a result of the real-world cases that we used in our research.

Recreate Methods

Which recreate methods and corresponding settings are needed, such that Adaptive Large Neighborhood Search performs the best?

The tests in Section 8.5 showed that using three recreate methods in a roulette wheel benefits the search. The solutions that were found had lower KPIs than in case only one recreate method was used. In case one recreate method was used, cheapest insertion found less improvements than parallel cheapest insertion and parallel regret insertion. However,

the differences in KPIs were small, and the recreate method did not seem to have a big influence on the quality of the solutions. This is in line with the results from the literature. Note that in case very large neighborhoods are used, the recreate method is expected to have influence, because reinserting the removed transports and finding improvements will be difficult.

Local Search

When and where are local search steps needed in the Adaptive Large Neighborhood Search heuristic, such that Adaptive Large Neighborhood Search performs the best?

Section 8.2 showed that local search steps were needed to plan all the transports for some of the instances. Therefore, local search was included in the construction phase of the heuristic. Furthermore, local search was used to intensify the search by temporarily accepting solutions for which the KPIs were at most 5% higher than the KPIs for the best solution that was found so far. Adding local search to intensify the search did decrease all KPIs at the cost of the computing time.

Main research question

To conclude, we answer the main research question. *In what way can the Adaptive Large Neighborhood Search heuristic be configured to find a near-optimal solution for a rich and real-world VRP, such that the computing time needed is acceptable for the instance that is looked into?*

The findings that were reported on in Section 8.2 up to, and including, Section 8.5 resulted in two configurations that needed extra tuning for the number of transports that were removed during the search. The results of these additional tests, that were reported on in Section 8.6, contributed to the two final configurations that are used to answer the main question.

The components that together form a hybrid method, called ORTEC Adaptive Large Neighborhood Search, that found solutions with the lowest KPIs, consist of five removal methods (related removal, random removal, worst removal, cluster random removal and cluster worst removal) and three recreate methods (cheapest insertion, parallel cheapest insertion and parallel regret insertion) combined in a roulette wheel, for a total of 1200 iterations, where the number of transports that is removed increases as follows: 5%, 8%, 12%, 15%, 18% and 22%. For instances B, the KPIs improved with percentages between 3.10% and 5.69%, compared to the original customer configuration. The solutions were found within 37 minutes, which is an acceptable computing time for the customer. For the instances C, the KPIs improved with percentages between 1.79% and 2.00% and the solutions was found within 27 minutes, compared to the initial solution.

In order to reduce the computing time, we provide a heuristic, called ORTEC Large Neighborhood Search, that found solutions with the lowest KPIs in case only one removal and one create method are used. This heuristic consist of related removal as removal method in combination with parallel cheapest insertion to recreate the solution, for a total of 500 iterations, where the fixed number of transports that is removed is equal to 10%. The KPIs improved with percentages between 0.55% and 4.06% for the instances B, compared to the original customer configuration, and the solution was found within 17 minutes. The hours did increase with 1.07%, however this is the result after the cost set that is used as input is adjusted. Comparing the results with the cost set that is used by the customer, all KPIs improved. For the instances C, the KPIs improved with percentages between 1.36% and 1.77%, compared to the initial solution, and the solutions were found within 12 minutes.

9.1 Recommendations for further research

In order to proceed this research, the following recommendations are provided for further research.

Our research included tests regarding the construction of the initial solution and the influence of it during the search. We speculated that the quality of the initial solution influenced the quality of the solution found with removing and reinserting transports, especially in case only one removal and one recreate method are used. Further research needs to be done, to conclude whether solutions of good quality are needed for LNS. It seems to be the case that the LNS search process converges faster to a near-optimal solution with an initial solution of good quality.

Furthermore, minor differences are found in case a different solution is used as starting point for removing and reinserting transports with ALNS. For some instances, the ALNS with a roulette wheel that was reported on in this thesis, was able to find the lowest KPIs with simple insertion methods, such as parallel regret insertion, that found initial solutions with the highest KPIs. It would be interesting to see whether initial solutions that do not already have a tight planning, result in solutions that are closer to the optimal solutions, when ALNS is used, than the case were an initial solution of good quality is used. We were not able to investigate this, because the simple insertion methods were not able to plan all transports for our instances. This could be avoided by using insertion methods during the search for improvements, in order to plan transports that were left unplanned during the construction of the initial solution.

Related removal (SR) and cluster random removal (CRR) are removal methods that have a good performance within the LNS we investigated. In the literature that we studied for this research, the performance of CRR is not mentioned in combination with LNS. Because which one of the two removal methods resulted in the lowest KPIs was case specific in our research, we recommend to investigate this.

We did not include comparison of the results with respect to the size and characteristics of the cases. Our results show that for smaller cases, the removal strategy has less impact on the performance of ALNS. The difference in KPIs for the number of removals that is used, and whether a roulette wheel is included is small. This might be explained with the fact that the number of possibilities of moving transports around is usually less for smaller cases. Less diversification might be needed to find improvements.

In what way the local search methods are used to intensify the search, should be investigated more. Including local search methods in the ALNS framework, did improve all KPIs for our cases. There are many local search methods known in the literature, and hence many variants can be investigated. Furthermore, the computing time might decrease by having other local search methods, or different configurations for the local search methods that we included. We work with an intensification phase of 100 iterations. However, it can be the case that improvements are already found after less iterations, and this number could be reduced.

Unfortunately, we were not able to provide figures that show the improvements that are found over time. Reducing the computing time may be easy, for example in case the last iterations do not provide improvements. Therefore, we recommend to investigate the performance of the heuristic over time.

At last, we want to recommend further research on the number of transports that are removed, compared with different removal methods. One can imagine that random removal needs more transports that are removed to find improvements, otherwise the removed transports are scattered over the network and will be reinserted in the same place. However, with for example related removal, also a smaller amount of transports that are removed and reinserted could provide improvements.

9.2 Recommendations for ORTEC

The recommendations given in the previous section are relevant for all studies that involve large neighborhoods and real-world cases. We also provide recommendations for further research that is related to ORTEC.

The method for temporarily accepting solutions that have at most $p\%$ higher KPIs, needs further research. Our tests included 1 and 5 as values for p. Increasing this number increases the computing time, because more solutions will be temporarily accepted and thus local search steps are more often executed. However, this might lead to solutions that can not be found with lower values for p, provided that the intensify method is strong enough to find improvements. Tuning this parameter p in combination with tests including different local search methods, may lead to a faster convergence to a near-optimal solution.

Furthermore, we advise to include some randomization in two removal methods: worst removal and cluster worst removal. In case these methods do not find improvements, and the removal methods are executed several times in a row, the same transports will be removed. The transports are already ordered based on the additional costs of including the transport in the solution. We recommend to include randomness in choosing which transport is seen as worst inserted transport and is removed from the solution.

To conclude, further research needs to be done with respect to the computing time of the roulette wheel that is included in our ALNS. Including a learning layer with the use of a roulette wheel, instead of a recursion of the removal and reinsertion methods, results in extra computing time.

Appendix A

ORTEC software

The research that is reported on in this thesis uses software that is developed by ORTEC, called *ORTEC Routing and Dispatch* (ORD). The software is used to plan transports into routes and to assign possible resources to these routes. To provide more information on how the users see and use this software, this appendix is included.

One can choose to plan the transport on routes manually, but there is also a possibility to use the automated planning process.

To illustrate the usage of ORD, a screen shot of the software is included in Figure A.1. Transports and routes are shown in grids. It is possible to give different colors to, for example, transports that are planned or not, and routes that are empty or have transports planned on them. In this way, it is easy to read the grids and select transports and routes for the automated planning. In Figure A.1 also another grid can be seen on the right. When clicking on a route, this grid will be filled with information of this route. The information differs per customer, but it always shows the actions in that route. Actions include couple and decouple of the truck to the trailer, pickup and delivery tasks, stops and travels. In case time windows are included, and a vehicle arrives before the start of the time interval, a wait task is shown in this grid as well. Furthermore, if the drivers legislation is taken into account, also breaks are scheduled and shown in this grid. All actions can be customized with colors and icons that are shown in the grid as well. For example, in the action grid on the right in Figure A.1, a time violation is visualized. The warning of a violation, of any kind, is represented with a triangle and exclamation mark. The type of violation is given next to it with an icon. For the time violation in this case, a small clock is used.

As for the transports and routes, the grids where they are shown can be customized as well. Several columns can be added, when the data allows it. Without the right information in the data, the column will be left empty. For the transport grid, one can add for example the columns with the transport number, volume, the time of possible pickup and delivery, the distance between pickup and delivery and the transport date. The grid for the routes contains columns as well. For example, where the route starts and ends, the start time, the number of stops, the name of the driver, total volume and whether the route is finished or not.

Furthermore, in ORD it is possible to use filters in the grid which can be adapted to the needs of the planner. One can filter based on (multiple) columns that are visualized in the grid. For both the transports and routes there is a comment column available to make short notes involving the transports and routes. Also a map is available where all the locations in the network can be plotted. It is even possible to visualize the routes, or a couple of routes, and include the stops as well. This makes the planning process easier for the planners, as they are able to visualize the routes they are creating.

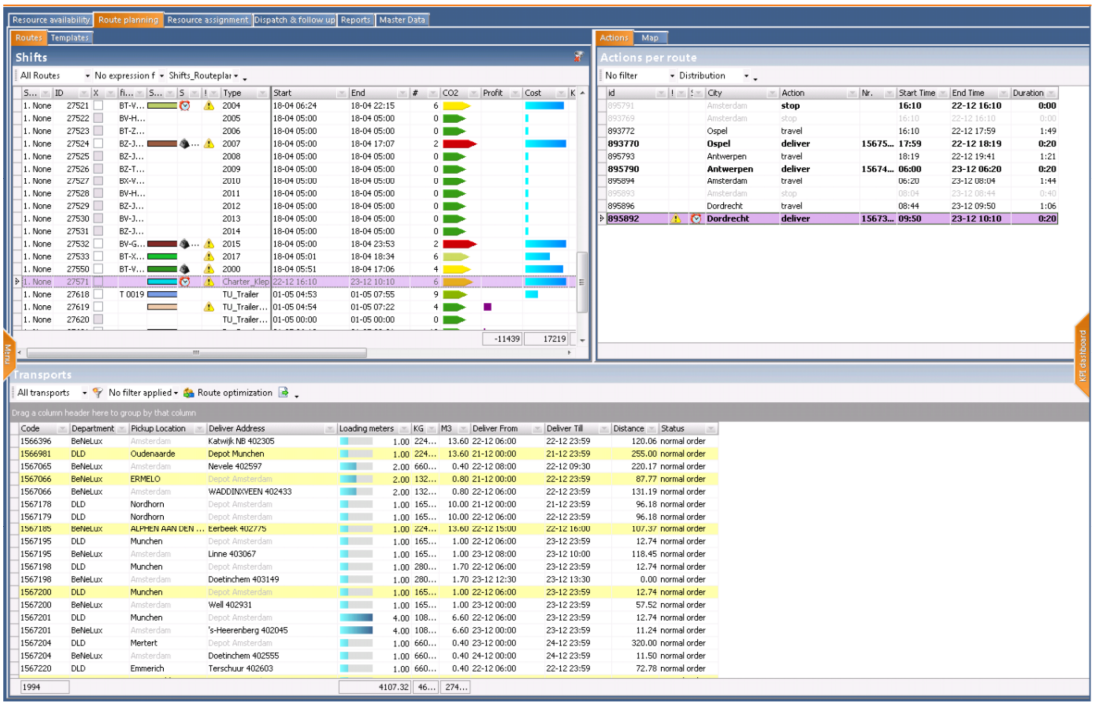


Figure A.1: An example of a layout for route planning in ORD. There are three grids shown in this figure: top left is the route grid, top right the action grid and at the bottom is the transport grid.

Appendix B

Extended Literature Study

B.1 Initial Solution - Performance

There are various construction methods for an initial solution described in the literature. From simple heuristics to more sophisticated ones. Random insertion is a method that is very simple and does not need a lot of configuration. Jaszkiewicz and Kominek (2003) and Lin (1965) use this method to construct an initial solution that is used as input for their heuristic. However, when small neighborhoods are searched, a random initial solution can slow down the process of finding solutions of good quality. Therefore, one could advocate starting with an initial solution of good quality. Jaszkiewicz and Kominek (2003) mention that the efficiency of their heuristic could be increased when the initial solution would be of higher quality. They propose to use a simple, but specialized heuristic.

An insertion method that takes into account the characteristics of the transports and the routes, is cheapest insertion. Azi et al. (2010) extend the cheapest insertion heuristic by allowing a route to be split into two routes, when a transport can not be planned on any of the routes. They are able to do this, because vehicles are allowed to drive several routes in the VRP with Multiple Routes, which they investigate. Schrimpf et al. (2000) use an additional vehicle in case a transport can not be inserted, due to capacity or time window constraints. To show the power of this insertion method, compared to random insertion, they made 100,000 solutions with both random insertion and cheapest insertion. This resulted in two distributions, where the average solution length of solutions produced with cheapest insertion was much lower than the average length of the solutions created with random insertion. The length of the solution being defined as the sum of all driving distances in the network. For cheapest insertion, the average length was only 15% above the optimal solution, and the average length was only 8% of the average length of the solutions created by random insertion. Schrimpf et al. (2000) stat that because of these good initial solutions, their method converges faster than when random insertion is used.

There is also a method provided to minimize the number of vehicles that is used, before using an ALNS framework. This method is proposed by Ropke and Pisinger (2006a). They first build an initial feasible solution, using sequential insertion. After that, they try to minimize the number of vehicles that are used, by removing one route at a time. The transports on that route are reinserted, using a LNS heuristic. In case all transports are inserted, they stop the LNS heuristic and move on to another route that is removed from the solution. In case the LNS heuristic is not able to find a solution where all transports are planned again, the search for a better solution with less vehicles is stopped. The resulting solution will be used as the initial solution for the ALNS framework. One can imagine that this can increase the computation time by a lot, and therefore they use a maximum number of iterations that can be used for the vehicle minimization part. They also propose another method to keep the computing time limited. For this second method, the LNS heuristic is stopped in case five or more transports are unplanned and there are no improvements found in the last 2,000 iterations.

Azi et al. (2010) are the only one, as far as the author's knowledge goes, that mention a method that copes with an infeasible initial solution. The unplanned transports, that make the initial solution infeasible, may be planned using a recreate method. Therefore, the recreate method will not only insert removed transports, but all transports that are not planned at the moment of executing the recreate method. In this way, the transports that were not planned during the construction of the initial solution, may be planned during this recreate stage.

B.2 Removal Methods

Several removal methods are used in metaheuristics that use the idea of removing and reinserting transports. This appendix is included to elaborate on some of the removal methods that can be included in ALNS.

Related Removal

Shaw (1997) proposed to use a method to remove transports that are similar, where the similarity is measured with a relatedness function. This function includes measures for the distance between the delivery locations of transports, the similarity of the time windows on the transports, whether they are scheduled on the same route and whether the transports have similar load weights. A pseudo-code on how to configure the use of this relatedness function is included in Algorithm 7.

Algorithm 7: Related Removal (SR)

Input: An initial solution s. T are all transports in the network, T_r is the set of transports that are removed, t_n is the number of transports that need to be removed, L contains all transports that are not in T_r, p controls the randomness in the algorithm.

begin

1 Select a transport t from the solution s at random
2 T_r={t}
3 **while** $|T_r| < t_n$ **do**
4 Select $t \in T_r$ at random
5 Sort L according to their similarity to t
6 Choose a random number $r \in [0, 1)$
7 Select transport j that is at location $r^p|L|$ in L
8 $T_r = T_r \cup \{j\}$
end while
9 Remove the transports that are in T_r from the solution s

end

From this pseudo-code can be seen that a randomization is included. In case an initial transport is chosen several times during the search, this randomization will prevent that the same related transports are chosen to be removed. As can be seen in the pseudo-code, during each iteration of choosing transports to be removed, a transport t is chosen at random from the already removed set of transports. Then, a random number r between 0 and 1 is chosen and all transports that are not yet removed are ordered on similarity to transport t. At last, a transport from this ordering is chosen to be removed, based on the randomization number r and a diversification parameter p. In case this p is 0, the next transport to be removed is chosen at random from the ordering. In case p equals ∞, the most related transport is chosen from the ordering.

Ropke and Pisinger (2006a) extend the relatedness function in such a way, that both pickup and delivery locations and times are included. This is needed in case there are multiple depots for example. They also change the boolean variable that indicates whether

two transports are scheduled on the same vehicle or not. In their relatedness function it is ensured that two transports only obtain a high value for relatedness in case they can only be scheduled on a few vehicles. For more details on this relatedness variant is referred to Ropke and Pisinger (2006a). This relatedness function is also used by Ropke and Pisinger (2006b) and Pisinger and Ropke (2007).

Another related removal method that is based on Shaw (1997), comes from Azi et al. (2010). They only account for the geographical distance, d_{ij} in (6.1), and the absolute difference between the time of beginning of service at both locations, $|t_i - t_j|$ in (6.1). These two measures are weighted with parameters that need to be specified. Also the parameter p is used to control the degree of relatedness, the same way it is used by Shaw. Azi et al. (2010) used a value of 6, instead of 4 that Shaw used for his test. The relatedness is considered on a customer or transport level, but they also describe the relatedness on route level. To do so, they propose two different measures for the relatedness between routes. One measure takes into account the distance between the gravity centers of two routes, which is the distance between the average location over all customer locations in both routes. The other measure looks at the smallest distance between any pair of customers taken from the two routes that are considered. Azi et al. (2010) tested these measures separately, but a combination with weight parameters is tested as well. Based on tests on the Solomon instances, 11 instances for R2, 8 instances for C2 and 8 instances for RC2, they found that the minimal distance measure performs better than the gravity center-based measure for route level. Besides that, they also found that on route level using one measure performs better than weighing two measures.

Historical-based Removal

Ropke and Pisinger (2006b) propose two versions of a historical-based removal method. The first method is called *neighbor graph removal*. This method uses a complete, directed, weighted graph: the neighbor graph. All visits in the network have a node in this neighbor graph and all arcs have an initial weight of infinity. After each new found best solution, the weights of the arcs in the neighbor graph are updated and the cost of the best solution is stored. The second method is called *request graph removal* and the historical information is stored in the request graph. This graph is complete and undirected, contrary to the directed neighbor graph, and consist of nodes that represent the request, or transports. The weight on an edge between two transports is the number of times they are served with the same vehicle, regarding a specific number of best unique solutions. In the tests done by Ropke and Pisinger, this number was set to 100, and the initial weight on the edges was set to 0. A transport i obtains a score based on the sum of the weights on the edges to the transports that are in the same route, at that moment, as the transport i. In case the score is low, this transport i has not been planned many times with the transports that are currently in the route with i, according to previously found best solutions. Therefore, good solutions that are already found, indicate that this transport i is not placed in a suitable route. Improvements may be found when this transport is removed from the solution and reinserted in some other place. However, Ropke and Pisinger found that this method does not perform well, because of the lack of diversification. The changes are based on the best found solutions so far, which may result in small changes only. Therefore, they propose to use this request graph for the relatedness of two request.

B.3 Removal Methods - Performance

There are some articles that tested several removal methods, which are summarized in Section 6.2. More information on the tests that are performed in the literature is given in this section. However, for most of the ALNS heuristics, the performance of the single removal methods is not given, but is rather looked at the performance of the heuristic itself.

Ropke and Pisinger (2006a) tested related, random and worst removal. The tests were performed on four of Li and Lim's benchmark problems and 12 randomly generated in-

stances. These 12 instances contain both single depot and multi depot problems, and problems where some tasks may only be scheduled on certain vehicle types, hence there is a heterogeneous fleet. The four instances from Li and Lim contain between 50 and 100 tasks and the randomly generated instances contain 50 tasks. Ropke and Pisinger state that for their instances the related removal provided the best solutions, worst removal second best and random removal provided the worst solutions. From this may be concluded that the two slightly more complicated removal methods perform best. Note that these conclusions are drawn based on the comparison of average gaps between the solutions and the best solutions found during all tests.

Ropke and Pisinger (2006a) also proposed the ALNS, and therefore they are able to use different methods during the same search. To do so, they use the roulette wheel principle, where weights are assigned to the removal methods based on their successes in previous iterations in the search. They give the progress of the weights for the removal methods, as they change during the search, for a randomly chosen instance. For that specific instance, worst removal obtained the lowest weights, whereas the weights for related and random removal followed quite the same progress during the search and obtained higher weights than worst removal. Note that there can not be drawn any conclusions from this, because this is only based on one instance. When the average progress over all instances is formulated, conclusions can be drawn. Furthermore, they state that some removals can be used to diversify the search and other can be used to intensify the search, as is explained in Chapter 4. Random removals may for example contribute to the diversification of the search, as they can contribute to large changes in the solution. However, when one wants to intensify the search, more critical tasks can be removed using worst and historical-based removal.

In Ropke and Pisinger (2006b) there are three different configuration that are tested:

- related, worst and random removal with roulette wheel selection,
- related, worst, random, cluster and historical-based removal with roulette wheel selection,
- related, worst, random, cluster and historical-based removal without roulette wheel selection.

They tested these three configurations on several problems. They mainly chose problems with backhauls and problems with multiple depots and mixed vehicles and had a total of 338 instances for 16 problems. On these instances, the configuration with three removal methods performs worse than the two configurations with more removal methods. The average gap, in percentage, for the three removal methods configuration is 0.81, with 201 best found solutions, whereas the average gap for more removal methods without and with roulette wheel selection is 0.62 (234 best solutions found) and 0.50 (248 best solutions found) respectively. As for the computation time, the time increases in case the number of backhaul customers increases. They explain this by the fact that routes are longer, thus there are many possibilities in the order of scheduling the tasks. Furthermore, the computation time increases with the problem size.

Furthermore, Azi et al. (2010) use removal on three different levels: customer level, route level and workday level. They first remove on workday level, random and related removal, followed by removals on route level, random and related removal, and last removals on customer level, only random removal. For each level, the removals are used for a few iterations, before continuing with the next level. They tested on Solomon instances with long horizons, as already mentioned in this thesis, and were able to improve the number of unplanned customers with 7.81%, 8.22% and 6.92% for classes RC2, R2 and C2, respectively.

B.4 Number of Removals - Performance

Besides the removal method itself, also the number of transports that are removed are needed as input for the metaheuristics that remove and reinsert transports.

For the number of tasks to be removed, Shaw (1997) starts initially with 25 tasks. For each recreate iteration that was not able to find improvements, the number to be removed decreased by one. On the other hand, in case the recreate method was able to find improvements for 20 consecutive iterations, the number was increased by one. He tested this procedure on the Solomon instances with a short scheduling horizon. For the C1 class, the heuristic found the best solution at the first output moment, which was after 7.5 minutes. From this Shaw concludes that the C1 class is rather easy to solve. However, for R1 and RC1, the convergence is much slower.

Schrimpf et al. (2000) test radial, random and sequential removal for 100 iterations on the TSP PCB442. For 442 transports, they test removals of 1%, 2%, 5%, 10%, 20% and 50% of the total number of nodes that are visited by in the TSP. From the graphs and tables they state, a few conclusions may be drawn. When the percentage of tasks to be removed is increased from 1% to 5%, the length of the solution decreases for all removal methods. The length of the solution is, in this case, the total distance of the circuit in the TSP. When the percentage of tasks to be removed is increased from 10% to 50%, whether the length of the solution decreases or increases, depends on the acceptance criteria that is used. For random walk acceptance, the solution length decreases for random removal, increases a bit for radial removal and a lot for sequential removal. For greedy acceptance, the behavior of the removal methods is different, because the length of the solution decreases for radial and random removal, and increases for sequential removal. For the test instance of Schrimpf et al. (2000) can be concluded that radial removal performs best with 5% and 20% for a random walk acceptance and greedy acceptance, respectively. However, the difference in solution quality between 1%, 2% and 5% is within 0.5%, and the same holds for 10%, 20% and 50%. For sequential removal similar behavior can be discovered: 2% works best for random walk acceptance and 20% for greedy acceptance. Again, the difference between 1%, 2% and 5% is within 0.5%, and the same holds for 10% and 20%. For random removal can be concluded that the highest percentage works best, so 50%, for both acceptance criteria. From this can be concluded that the number of removals, a small or high percentage, is heavily depending on both the removal method and acceptance criteria.

A few years after Schrimpf et al. (2000), Ropke and Pisinger (2006a) use a completely different method. They propose to use a method that randomly chooses a 'degree of destruction' in each iteration. This could be beneficial, because in this way still a lot of solutions can be explored and also a large range over the solution space can be visited. The choice for the number of removals is depending on the instance size. As instance size, Ropke and Pisinger take the number of requests n, so the number of transports that are in the network. They choose a random number in the interval $[4, \min(100, \xi n)]$, where ξ is a parameter that needs to be specified and control how many tasks are removed. For this ξ, Ropke and Pisinger tested values from 0.05 up to 0.5 with steps of 0.05. They conclude that with a low value of ξ, there are not many tasks removed and there is a higher chance of getting trapped at local minima. However, in case ξ is relatively large, it is hard for the insertion heuristics to find a good solution. A disadvantage that comes along with a high number of removals, is that each iteration will take longer, because more tasks are removed. Ropke and Pisinger found that $\xi = 0.4$ works best for their test instances, and therefore they choose a random number of tasks to be removed from $[4, 0.4n]$. From the test instances they used, four have a maximum of 100 tasks and 12 a maximum of 50 tasks.

In their next research, Ropke and Pisinger use the same amount of removals (Ropke & Pisinger, 2006b). However, the number of tasks to be inserted is in Ropke and Pisinger (2006b) for some instances higher than the maximum of 100 in Ropke and Pisinger (2006a). They conclude afterwards, that in case 100 tasks are removed this is too much (Pisinger & Ropke, 2007). They discovered that with many removals, the insertion techniques they used, cheapest and regret insertion, were too weak. That is why they changed the lower and upper bound for the interval they proposed in Ropke and Pisinger (2006a). The new interval is $\big[\min\{0.1n, 30\}, \min\{0.4n, 60\}\big]$, so for large n this will mean the interval [30, 60] is used. Therefore, the choice for the number of tasks that need to be removed lays between 30 and 60 for large instances.

Azi et al. (2010) use three levels for the removals, see Section 6.2.3. They test how many

iterations the heuristic has to stay at each level and what number of removals have to be chosen. They test this on Solomon's RC2 intance with 100, 400 and 800 customer locations. They tested 100, 200 and 400 iterations for each level and three different intervals for the degree of destruction, [5%, 35%], [35%, 65%] and [65%, 95%]. They use the same method as Ropke and Pisinger propose, hence they select a uniformly random number from the interval and use that as the percentage of tasks that need to be removed at the level that is considered. As others already concluded, in Azi et al. (2010) is also concluded that the computation time increases when the number of removals increases. Furthermore, they state that the solution quality, based on unplanned customers, total distance and computation time, decreases when considering higher lower and upper bounds for the interval for the degree of destruction. They conclude that for RC2 with 100, 400 and 800 customers, the performance of their heuristic is best when the number of removals is chosen from the interval [5%, 35%] and each level has 200 iterations of removals.

Appendix C

Additional Figures

Not only the construction method for the initial solution may influence the performance of the recreate method, but also the removal method that is used before the transports are reinserted. Therefore, the tests with different recreate methods are also executed with configurations that use SR as single removal method. The results are stated in Table C.1.

Table C.1: Results Recreate Method on SR (Average)

Recreate Method	Trips	Tasks	Plan Costs (€)	Distance (km)	Hours (h)	Driving Time (sec)	Run Time (h:min)
PCI	739	11,064	18,968	5304	169.57	320,984	2:03
PRI	737	11,064	18,978	5329	169.50	321,983	3:49
CI	741	11,064	19,035	5336	170.07	322,849	3:20

This table shows the results of having a fixed method for the recreate method, where related removal (SR) is used to remove transports. The different recreate methods are: parallel cheapest insertion (PCI), parallel regret insertion (PRI) and cheapest insertion (CI). The second and third column show the sum of the total amount of trips that are used and tasks that are planned, respectively. The maximum number of tasks to plan is $(6 \times 794) + (6 \times 794) + (6 \times 256) = 11,064$. The last five columns contain information about the other KPIs that are used to indicate the performance of the configurations.

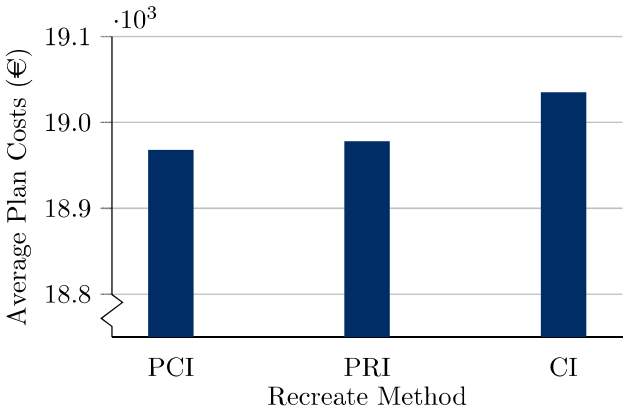
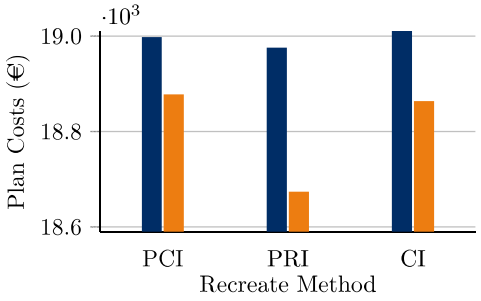
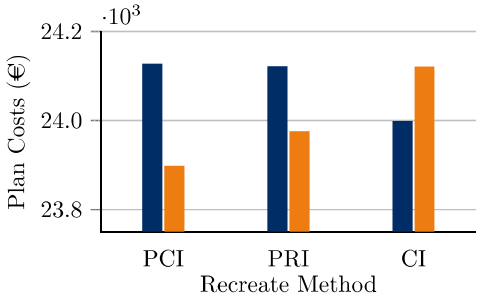


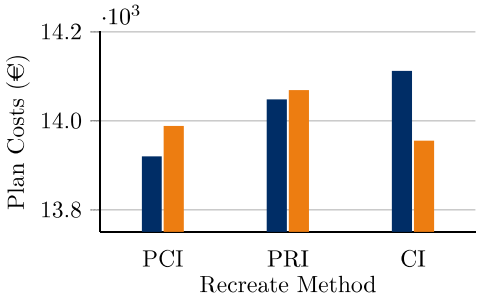
Figure C.1: The average plan costs, in euros, for three different fixed recreate methods: parallel cheapest insertion (PCI), parallel regret insertion (PRI) and cheapest insertion (CI). The average is taken over three instances, two methods for constructing the initial solution and three configurations, resulting in 18 different cases per recreate method. For the removal component, related removal (SR) is used as a single removal method. Note that the values on the y axis do not start at zero and the scale is in thousands.



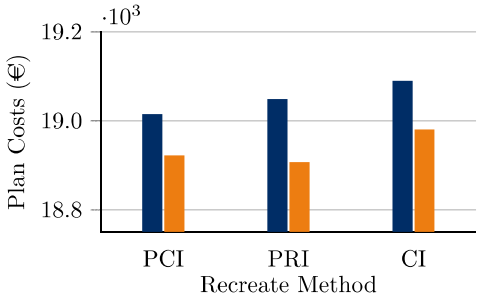
(a) The average plan costs (in euros) per recreate method for instance B1.



(b) The average plan costs (in euros) per recreate method for instance B3.



(c) The average plan costs (in euros) per recreate method for instance B4.



(d) The average plan costs (in euros) per recreate method.

Figure C.2: The average plan costs (in euros) for three instances and the average over all instances per recreate methods, split up into two configurations for the construction of the initial solution: sequential insertion (SI) and parallel insertion (PI). For the removal component, related removal (SR) is used as a single removal method. Note that the y axis are in thousands and the values do not start at zero.

Appendix D

Tables

The results that are used in figures in Chapter 8 are represented in tables in this appendix. For the instances B, we included figures per instance and initial solution, hence the tables show results per instance. The tables for instances C are split into two initial solutions: SI and PI, and the trips and tasks are summed over all instances C. The other KPIs show the average results over the instances C.

Table D.1: Results initial solution per case B

Configuration	Case	Trips	Tasks	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)	Run Time (h)
145	B1	121	2382	18134	5218	177.66	323754	2.56
145a	B1	122	2382	18068	5290	174.21	323795	2.91
145b	B1	121	2382	17986	5228	174.49	322048	2.32
146	B1	122	2382	18025	5286	173.46	323559	2.75
146_2	B1	124	2382	18028	5256	172.76	322138	4.05
145	B3	153	2382	23311	6451	196.06	380752	6.72
145a	B3	157	2382	23179	6310	194.45	370241	6.48
145b	B3	156	2382	23252	6349	195.25	373362	7.60
146	B3	155	2382	23347	6353	197.82	374463	7.48
146_2	B3	156	2382	23297	6396	194.95	376092	6.00
145	B4	89	768	13912	3911	115.76	239228	1.10
145a	B4	90	768	13841	3858	114.94	235997	1.15
145b	B4	91	768	13882	3847	115.03	235611	1.06
146	B4	89	762	13759	3847	114.58	237007	1.01
146_2	B4	92	768	13873	3844	114.45	236458	0.84

Table D.2: Results multiple removal methods per initial solution, per instance B

Removal Method	Initial Solution	Case	Trips	Tasks	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)	Run Time (h)
3RW	SI	B1	121	2382	18178	5269	177.51	324194	2.94
3RW	PI	B1	121	2382	18074	5314	174.53	326298	2.84
3RW	SI	B3	155	2382	23300	6444	194.93	379191	7.06
3RW	PI	B3	156	2382	23283	6415	194.51	378122	6.64
3RW	SI	B4	90	768	13985	3952	115.92	241350	0.57
3RW	PI	B4	92	768	13831	3821	114.08	235124	0.40
3noRW	SI	B1	122	2382	18704	5368	185.39	334149	2.23
3noRW	PI	B1	126	2382	18584	5338	180.92	331644	1.93
3noRW	SI	B3	155	2382	23602	6522	198.94	384178	5.71
3noRW	SI	B3	157	2382	23673	6480	199.80	382232	4.63
3noRW	PI	B4	87	768	13949	3946	116.86	241604	0.57
3noRW	SI	B4	94	768	13984	3855	114.92	237232	0.45
5RW	PI	B1	121	2382	18134	5218	177.66	323754	2.56
5RW	SI	B1	124	2382	18028	5256	172.76	322138	4.05
5RW	PI	B3	153	2382	23311	6451	196.06	380752	6.72
5RW	SI	B3	156	2382	23297	6396	194.95	376092	6.00
5RW	PI	B4	89	768	13912	3911	115.76	239228	1.10
5RW	SI	B4	92	768	13873	3844	114.45	236458	0.84
5noRW	SI	B1	121	2382	18354	5405	178.32	334396	2.51
5noRW	PI	B1	125	2382	18225	5259	175.98	327303	2.05
5noRW	SI	B3	155	2382	23304	6440	194.87	379264	5.21
5noRW	PI	B3	156	2382	23503	6416	198.34	380856	5.17
5noRW	SI	B4	89	768	13770	3840	114.54	235801	0.56
5noRW	PI	B4	93	768	13947	3851	114.97	237663	0.46
6RW	PRI	B1	120	2382	17891	5251	172.81	324850	2.37
6RW	SI	B1	119	2382	18611	5310	186.67	326642	2.65
6RW	PI	B1	122	2382	18175	5273	176.72	324048	2.54
6RW	PRI	B3	154	2362	23225	6433	194.58	378308	6.74
6RW	SI	B3	155	2382	23404	6444	196.93	379455	6.57
6RW	PI	B3	157	2382	23431	6412	196.58	377633	5.79
6RW	PRI	B4	88	758	13907	4028	114.78	244005	0.56
6RW	SI	B4	89	768	14017	3958	117.12	241842	0.67
6RW	PI	B4	89	768	13711	3802	114.04	234303	0.49

Table D.3: Results multiple removal methods per initial solution, instances C

Removal Method	Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
3RW	SI	6552	24888	5559515	51358	1962.61	3693285	49:10
3RW	PI	6556	24888	5560253	51348	1971.01	3691978	54:29
3noRW	SI	6577	24888	5576038	51502	1972.74	3703325	22:54
3noRW	PI	6573	24888	5577392	51513	1973.69	3703435	22:35
5RW	SI	6561	24888	5558084	51344	1962.10	3692100	48:36
5RW	PI	6556	24888	5557866	51327	1969.87	3690645	49:07
5noRW	SI	6576	24888	5581196	51558	1970.39	3706558	22:35
5noRW	PI	6575	24888	5582205	51561	1973.89	3706688	21:33
6RW	SI	6544	24888	5552683	51278	1968.43	3687302	48:14
6RW	PI	6563	24888	5561741	51368	1968.59	3694036	46:56

Table D.4: Results single removal methods per initial solution, per instance B

Removal Method	Case	Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)	True Time (h)
SR	B1	SI	120	2382	18904	5541	187.26	342225	2.18
SR	B1	PI	123	2382	18516	5349	181.35	331887	1.59
SR	B3	SI	155	2382	23909	6492	205.13	381885	5.98
SR	B3	PI	156	2382	23787	6526	201.75	383529	6.42
SR	B4	SI	90	768	14029	3941	116.75	241689	0.56
SR	B4	PI	93	768	13927	3854	114.59	236963	0.45
RR	B1	SI	121	2382	20321	5829	209.17	357050	2.78
RR	B1	PI	131	2382	19576	5547	193.24	341060	1.75
RR	B3	SI	156	2382	24885	6732	218.73	392792	4.77
RR	B3	PI	162	2382	24789	6598	216.10	385795	4.56
RR	B4	SI	90	768	14220	4069	117.83	247612	0.44
RR	B4	PI	96	768	14235	3940	116.44	242123	0.50
WR	B1	SI	118	2382	20951	5955	221.25	363470	2.98
WR	B1	PI	131	2382	19940	5686	197.74	346250	1.86
WR	B3	SI	156	2382	26105	6889	239.72	400922	0.59
WR	B3	PI	160	2382	25216	6538	227.18	384130	3.85
WR	B4	SI	90	768	14453	4142	120.95	249327	0.58
WR	B4	PI	98	768	14433	4044	116.85	244142	0.57
CRR	B1	SI	121	2382	19511	5657	196.41	347327	2.46
CRR	B1	PI	127	2382	18809	5406	183.39	332373	1.77
CRR	B3	SI	159	2382	24035	6550	204.56	383272	5.02
CRR	B3	PI	158	2382	23984	6449	205.95	378279	4.81
CRR	B4	SI	91	768	13937	3897	115.35	239210	0.56
CRR	B4	PI	94	768	14047	3874	115.45	238614	0.48
CWR	B1	SI	120	2382	20806	5950	217.13	360461	2.25
CWR	B1	PI	131	2382	19867	5615	197.71	344679	1.66
CWR	B3	SI	155	2382	26278	6916	243.16	402110	1.35
CWR	B3	PI	160	2382	25119	6541	225.30	382833	4.10
CWR	B4	SI	90	768	14452	4141	120.96	249346	0.52
CWR	B4	PI	98	768	14425	4026	117.05	243568	0.50

Table D.5: Results single removal methods per initial solution, instances C

Removal Method	Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
SR	SI	6563	24888	5581116	51547	1975.84	3705648	46:32
SR	PI	6564	24888	5588012	51614	1976.68	3709335	46:44
RR	SI	6600	24888	5592129	51656	1975.33	3713787	32:13
RR	PI	6593	24888	5588041	51613	1976.89	3710553	31:38
WR	SI	6661	24888	5651391	52202	1997.58	3749178	17:10
WR	PI	6676	24888	5668190	52350	2007.10	3757388	16:22
CRR	SI	6571	24888	5579371	51534	1973.28	3704834	45:21
CRR	PI	6582	24888	5582436	51557	1976.76	3706609	43:09
CWR	SI	6660	24888	5655537	52238	2000.17	3751290	19:39
CWR	PI	6676	24888	5670206	52368	2007.86	3759226	18:59

Table D.6: Results number of removals per initial solution, per instance B

Percentage	Initial Solution	Case	Trips	Tasks	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)	Run Time (min)
1	SI	B1	41	794	19687	5513	201.47	340620	6
1	PI	B1	42	794	19106	5480	188.53	337191	4
5	SI	B1	41	794	18277	5349	176.56	329227	12
5	PI	B1	41	794	18155	5346	174.19	328012	10
10	SI	B1	41	794	18153	5271	175.65	324663	15
10	PI	B1	43	794	18434	5248	177.73	323110	12
20	SI	B1	41	794	19418	5452	197.32	338425	28
20	PI	B1	42	794	18847	5343	186.08	328118	21
30	SI	B1	42	794	18511	5317	179.88	326852	43
30	PI	B1	41	794	18513	5335	181.56	330373	30
40	SI	B1	43	794	19496	5541	193.10	337551	61
40	PI	B1	41	794	19366	5566	193.99	342757	58
50	SI	B1	42	794	18704	5293	184.21	328618	76
50	PI	B1	42	794	19223	5416	192.12	333843	78
1	SI	B3	51	794	24617	6640	217.54	387466	12
1	PI	B3	53	794	24975	6514	223.61	381063	9
5	SI	B3	52	794	23493	6483	197.59	380970	23
5	PI	B3	52	794	23786	6319	206.56	373160	19
10	SI	B3	53	794	23897	6538	201.59	384953	32
10	PI	B3	52	794	23915	6471	206.07	381947	30
20	SI	B3	52	794	24281	6468	212.67	383428	57
20	PI	B3	53	794	24144	6526	206.36	382704	60
30	SI	B3	55	794	24523	6468	211.69	377424	77
30	PI	B3	53	794	24088	6386	208.45	375583	94
40	SI	B3	53	794	24771	6453	221.37	378817	89
40	PI	B3	53	794	23776	6348	203.57	372906	113
50	SI	B3	53	794	24950	6659	220.42	390175	136
50	PI	B3	53	794	25396	6556	231.01	384731	132
1	SI	B4	30	256	14193	4070	117.26	245553	1
1	PI	B4	32	256	14247	3963	116.09	241411	1
5	SI	B4	31	256	13884	3834	114.20	235425	2
5	PI	B4	31	256	14017	3887	115.60	239787	2
10	SI	B4	29	256	13872	3944	115.94	241279	3
10	PI	B4	30	256	13729	3805	113.89	234817	3
20	SI	B4	30	256	14042	3961	116.62	241570	5
20	PI	B4	31	256	13832	3803	113.78	233949	4
30	SI	B4	31	256	13860	3818	114.02	235575	5
30	PI	B4	31	256	13996	3859	115.54	237166	5
40	SI	B4	30	256	14064	3939	117.31	242541	8
40	PI	B4	30	256	13813	3850	114.46	236231	7
50	SI	B4	31	256	14128	3949	116.78	240933	8
50	PI	B4	30	256	14103	3950	118.24	242790	9

Table D.7: Results number of removals per initial solution, instances C

Percentage	Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
1	SI	2218	8296	5650128	52187	1999	3749653	03:06
1	PI	2223	8296	5658608	52255	2007	3753307	02:39
5	SI	2198	8296	5587354	51601	1979	3710049	09:12
5	PI	2198	8296	5600823	51727	1984	3718219	09:16
10	SI	2190	8296	5580211	51536	1976	3704560	11:07
10	PI	2189	8296	5595666	51678	1983	3714708	11:02
20	SI	2193	8296	5593082	51672	1973	3713835	11:32
20	PI	2197	8296	5598034	51695	1986	3714617	10:28
30	SI	2192	8296	5604483	51775	1978	3720752	11:36
30	PI	2202	8296	5612182	51830	1989	3724640	10:40
40	SI	2196	8296	5610652	51832	1980	3724503	14:24
40	PI	2207	8296	5616688	51866	1993	3726050	13:29
50	SI	2207	8296	5624718	51942	1995	3732201	16:37
50	PI	2212	8296	5626750	51957	1997	3732147	15:56

Table D.8: Results recreate method per initial solution, per instance B

Recreate Method	Initial Solution	Case	Trips	Tasks	Plan Costs (€)	Distance (km)	Hour (h)	Driving Time (sec)	Run Time (min)
PCI	SI	B1	124	2382	18159	5234	175.83	322956	5:32
PCI	PI	B1	126	2382	18271	5259	176.25	322444	4:17
PCI	SI	B3	155	2382	23314	6412	195.77	377740	3:58
PCI	PI	B3	157	2382	23573	6410	199.86	376881	3:10
PCI	SI	B4	87	768	13943	3955	116.75	241505	1:02
PCI	PI	B4	93	768	13940	3859	114.69	236916	0:32
PRI	SI	B1	123	2382	18229	5275	177.08	324802	6:16
PRI	PI	B1	124	2382	18316	5341	176.84	327758	4:51
PRI	SI	B3	155	2382	23128	6369	192.99	375009	4:25
PRI	PI	B3	156	2382	23343	6388	196.04	375889	4:47
PRI	SI	B4	89	768	13924	3924	115.93	239145	0:57
PRI	PI	B4	93	768	13954	3865	114.91	237202	0:40
CI	SI	B1	123	2382	18258	5268	177.81	323791	1:54
CI	PI	B1	123	2382	18389	5279	180.19	325460	1:35
CI	SI	B3	156	2382	23328	6437	194.82	378262	4:02
CI	PI	B3	157	2382	23466	6409	197.46	376743	3:09
CI	SI	B4	91	768	13905	3882	115.13	237443	1:01
CI	PI	B4	91	768	13854	3847	114.66	236790	1:01

Table D.9: Results recreate methods per initial solution, instances C

Recreate Method	Initial Solution	Trips	Tasks	Plan Costs (€)	Distance (miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
PCI	SI	6550	24888	556060351361		1966.37	3693778	49:20
PCI	PI	6553	24888	556029451352		1969.26	3692008	49:30
PRI	SI	6563	24888	556389851396		1965.36	3695762	46:56
PRI	PI	6556	24888	556522751390		1974.91	3695506	44:45
CI	SI	6555	24888	556164351376		1964.21	3694437	42:47
CI	PI	6560	24888	556462551394		1970.21	3695193	46:00

Table D.10: Results Percentage to Remove (Average)

Percentage	Initial Solution	Case	Trips	Tasks	Plan Costs (€)	Distance (km/ miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
5,12,20	SI	B	121	1844	18472	5215	163.07	314569	26:45
5,. . . ,22	SI	B	123	1844	18336	5147	160.51	310450	34:16
5,12,20	PI	B	123	1844	18206	5127	158.19	310140	23:49
5,. . . ,22	PI	B	123	1844	18317	5155	159.91	312618	39:00
20,12,5	SI	B	123	1844	18655	5152	166.95	311449	25:24
22,. . . ,5	SI	B	124	1844	18405	5141	161.33	310228	38:20
20,12,5	PI	B	125	1844	18672	5203	164.85	313161	24:34
22,. . . ,5	PI	B	125	1844	18459	5161	161.31	311403	33:45
5,12,20	SI	C	2188	8296	5565472	51408	1966.90	3695836	21:37
5,. . . ,22	SI	C	2189	8296	5558667	51360	1956.85	3693667	26:52
5,12,20	PI	C	2189	8296	5567790	51416	1974.80	3696332	20:32
5,. . . ,22	PI	C	2184	8296	5562136	51369	1970.34	3694332	25:17
20,12,5	SI	C	2189	8296	5562039	51377	1965.30	3694634	23:25
22,. . . ,5	SI	C	2187	8296	5565041	51409	1964.61	3696065	28:13
20,12,5	PI	C	2187	8296	5572830	51460	1977.97	3700227	22:15
22,. . . ,5	PI	C	2184	8296	5572742	51465	1975.12	3700707	26:36

Table D.11: Results Percentage to Remove Simple LNS (Average)

Percentage	Initial Solution	Case	Trips	Tasks	Plan Costs (€)	Distance (km/ miles)	Hour (h)	Driving Time (sec)	Run Time (min:sec)
10	SI	B	120	1844	18798	5287	168.62	318160	18:01
10	PI	B	122	1844	18822	5192	169.86	314193	15:41
10,20	SI	B	121	1844	19074	5304	173.26	320167	25:10
10,20	PI	B	124	1844	18981	5246	170.56	316927	22:39
10	SI	C	2195	8296	5585804	51600	1972.45	3709103	12:20
10	PI	C	2186	8296	5592203	51657	1976.29	3711048	11:19
10,20	SI	C	2193	8296	5581721	51559	1972.41	3707215	14:39
10,20	PI	C	2188	8296	5588297	51610	1980.31	3709575	13:53

References

Acharya, S. (2013). Vehicle Routing and Scheduling Problems with Time Window Constraints - Optimization Based Models. *Int Jr. of Mathematical Sciences & Applications*, *3*(1).

Ahuja, R., Ergun, O., Orlin, J. B., & Punnen, A. P. (2002). A Survey of Very Large-Scale Neighborhood Search Techniques. *Discrete Applied Mathematics*, *123*, 75–102.

Arora, S., & Barak, B. (2009). *Computational Complexity*. Cambridge University Press. Retrieved from `https://ebookcentral-proquest-com.tudelft.idm.oclc.org/lib/delft/detail.action?docID=433029`.

Azi, N., Gendreau, M., & Potvin, J.-Y. (2010). An Adaptive Large Neighborhood Search for a Vehicle Routing Problem with Multiple Trips. *CIRRELT*.

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, *35*(3), 268–308.

Bräysy, O. (2003). A Reactive Variable Neighborhood Search for the Vehicle Routing Problem with Time Windows. *INFORMS Journal on Computing*, *15*(4), 347–368.

Castro Martings, T. d., & Sales Guerra Tsuzuki, M. d. (2014). *Simulated Annealing: Strategies, Potential Uses and Advantages*. Hauppauge, NY: Nova Science Publishers, Inc.

Cook, J. W., Cunningham, W. H., Pulleyblank, W. R., & Schrijver, A. (1998). *Combinatorial Optimization*. New York, NY: John Wiley & Sons, Inc.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, *6*(1), 80–91.

El-Sherbeny, N. (2010). Vehicle Routing with Time Windows: An Overview of Exact, Heuristic and Metaheuristic Methods. *Journal of King Saud University*, *22*, 123–131.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: a guide to the theory of NP-completeness*. New York, NY: Freeman.

Glover, F., & Laguna, M. (1997). Tabu Search. *Kluwer Academic Publishers*.

Golden, B., Raghavan, S., & Wasil, E. (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges*. New York, NY: Springer Science+Business Media.

Hansen, P., & Mladenovic, N. (2001). Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, *130*(4), 449–467.

Häll, C., & Peterson, A. (2013). Improving Paratransit Scheduling using Ruin and Recreate Methods. *Transportation Planning and Technology*, *36*(4), 377–393.

Jaszkiewicz, A., & Kominek, P. (2003). Genetic Local Search with Distance Preserving

Recombination Operator for a Vehicle Routing Problem. *European Journal of Operational Research*, *151*, 352–364.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680.

Lin, S. (1965). Computer Solutions of the Traveling Salesman Problem. *BSTJ*, *44*(10).

Nagy, G., Wassan, N., Speranza, G., & Archetti, C. (2015). The Vehicle Routing Problem with Devisible Deliveries and Pickups. *Transportation Science*, *49*, 271–294.

Papadimitriou, C. H., & Steiglitz, S. (1998). *Combinatorial Optimization: Algorithms and Complexity.* Mineola, NY: Dover Publications, Inc.

Pisinger, D., & Ropke, S. (2007). A General Heuristic for Vehicle Routing Problems. *Computers & Operations Research*, *34*, 2403–2435.

Potvin, J.-Y., & Rousseau, J.-M. (1993). A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows. *European Journal of Operational Research*, *66*, 331–340.

Ropke, S., & Pisinger, D. (2006a). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, *40*(4), 455–472.

Ropke, S., & Pisinger, D. (2006b). A Unified Heuristic for a Large Class of Vehicle Routing Problems with Backhauls. *European Journal of Operational Research*, *171*, 750–775.

Schneider, J., & Kirkpatrick, S. (2006). *Stochastic Optimization.* Germany: Springer-Verlag Berlin Heidelberg.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record Breaking Optimization Results Using the Ruin and Recreate Principle. *Journal of Computational Physics*, *159*, 139–171.

Shaw, P. (1997). *A New Local Search Algorithm Providing High Quality Solutions to Vehicle Routing Problems* (Tech. Rep.). Glasgow: University of Strathclyde.

Sol, M., & Savelsbergh, M. W. P. (1992). *The General Pickup and Delivery Problem.* Retrieved from `https://pure.tue.nl/ws/files/2234894/389349.pdf`

Solomon, M. M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, *35*(2), 254–265.

Talbi, E.-G. (2013). *Hybrid Metaheuristics.* Berlin: Springer.

Toth, P., & Vigo, D. (2003). The Granular Tabu Search andd Its Application to the Vehicle-Routing Problem. *INFORMS Journal on Computing*, *15*(4), 333–346.

Toth, P., & Vigo, D. (2014). *Vehicle Routing: Problems, Methods and Applications* (2nd ed.). Philadelphia, PA: Society for Industrial and Applied Mathematics.