Erasmus University Rotterdam

Master's thesis Operations Research and Quantitative Logistics

New ways to handle sourcing volume constraints in vehicle routing problems

Author:
John Brouwer
343053jb

October 15, 2014

Supervisor EUR:
Dr. Remy Spliet
Co-reader:
Dr. Dennis Huisman

Supervisor ORTEC:
Ir. Wouter Terra

Abstract

In this thesis we develop methods to incorporate sourcing volume constraints in vehicle routing problems. These sourcing volume constraints can be useful when companies have made agreements with depots about the sourced volume of a specific product or when a depot is running out of stock. The constraints impose a minimum and maximum value on the volume of a specific product that can be sourced from a particular depot. The methods that we develop are designed such that they can be applied to large problem instances of the oil, gas and chemical industry.

Because it may be very time intensive to change optimisation algorithms of existing vehicle routing packages, we create methods that assign customers to depots before the routing algorithm is run (cluster first-route second). As we assume that the routing algorithm can solve multi-depot vehicle routing problems, we allow every customer to be assigned to multiple depots. In the first stage of the assignment phase we minimise the approximated cost of assigning every customer to one depot. We formulate a mixed integer linear program (MILP) for this problem, develop an urgency-based heuristic, and create a clustering algorithm to decrease the size of the MILP instances. The second stage of the assignment phase maximises the utility of assigning customers to additional depots without violating any constraints. To this end we formulate another MILP and create a greedy heuristic.

We develop various variants of variable neighbourhood search (VNS) heuristics that incorporate the sourcing volume constraints in the optimisation process. In the variable neighbourhood search 10 different neighbourhoods are explored. After an iteration of exploring neighbourhoods, we destruct and repair part of the solution to provide proper diversification.

The results show that solving the single-depot assignment MILP (although it is NP-hard) can be done in a matter of seconds for instances of industry size. The exact method also outperforms the developed heuristics. The multi-depot assignment phase shows to be a critical step in getting competitive results when using cluster first-route second. The exact multi-depot assignment problem can be hard to solve and remarkably the greedy heuristic consistently gives better routing results. For the VNS heuristics we clearly see different results for different variants. Finally, the VNS heuristics (integrated approach) generally perform better than the cluster first-route second approach, but we consider the latter to still be competitive. From this we can also conclude that the assignment methods make the right choices.

Contents

Chapter 1

Introduction

Companies in the oil, gas and chemicals industry have to deal with a large amount of difficult planning and scheduling problems. One of these problems is the downstream secondary distribution, which consists of transporting products by truck from multiple depots to a large number of customers. Vehicle routing problems (VRP's) like these are despite their long history in operations research still very relevant. One reason is that solving cases of real-world sizes to optimality often takes very long. As transport costs are a large component of logistic costs, companies are very eager to implement any improvements in software that can solve VRP's. Furthermore, companies are always interested in increasing the number of constraints that are taken into account in routing software, as this makes the routing schedules more realistic and decreases the time needed to manually adapt the schedule to the company's needs.

In most industries, a realistic vehicle routing problem consists of the transportation of a set of products by a mixed fleet of trucks. Also, customers want to be serviced within a given time window. In the oil, gas and chemicals industry there are some industry-specific factors that make vehicle routing even more complex. Tank trucks, as opposed to other trucks, are compartmented and can therefore only carry a limited combination of products and volumes. Trucks generally have between three and six compartments. Furthermore, customers may have demand for multiple products and want these to be delivered within one visit. To be able to serve all customers, drivers have to do multiple trips per shift, with the loading task separating the trips. The above-mentioned constraints are accounted for when solving variants of the petrol station replenishment problem described in literature (e.g. Cornillier et al. (2012)).

The additional constraint that we focus on in this thesis is a restriction on the volume of a specific product sourced from a specific depot. Sourcing volume constraints have not yet been considered in literature on petrol station replenishment, but are very relevant in this industry as many companies have contractual obligations with depots of competitors. Such a contract usually means that the company cannot source more than a certain volume of a specific product from the depot of that competitor per month or year. This strategic constraint can of course be translated to an operational constraint that has to be satisfied in the vehicle routing problem. Furthermore, it is not uncommon that depots almost reach a stock-out. In those cases there is a limited availability of some product at that depot. It is also possible that a company wants to source a minimum amount from a certain depot to apply for a discount. Regardless of the reason, the company decides per day what the minimum and maximum sourced volume should be for the different products at the various depots.

In this thesis we research how ORTEC[1] can implement this additional constraint in their routing software ORTEC Route Scheduling (see Appendix B for some information on the algorithms behind the software). Integrating the additional requirement of sourcing volume constraints in their algorithms is very time intensive because the core of ORTEC Route Scheduling assumes independence of vehicles. For example, in the optimisation process the assumption is made that changing the shift of one vehicle does not influence feasibility or costs of other shifts. When introducing sourcing volume constraints this no longer holds. A large part of the software would have to be rewritten to incorporate that trucks are not completely independent. Therefore, we add a pre-processing step where customers are assigned to depots. In the pre-processing one can ensure that the sourcing volume constraints are satisfied. On the other hand, splitting the optimisation in two separate steps (so-called cluster first-route second) is not ideal because in the routing phase it can turn out that different decisions should have been made in the assignment phase.

We research how we can develop this cluster first-route second approach in a real-world environment. This means that many different constraints hold and that the method should work on large cases within short running times. We analyse the different ways in which we can assign customers to depots. Finally, we show how competitive the cluster first-route second approach is by designing variable neighbourhood search heuristics and comparing the results.

In Chapter 2 we discuss literature on this research area. Chapter 3 formally describes the problem that we address. Then, in Chapter 4 we discuss the methods that we use to solve the described problem by a cluster first-route second approach. Following that, Chapter 5 shows our variable neighbourhood search approach to solve the problem in one step. Then, in Chapter 6 we describe and evaluate the experiments that we run to compare performance of the developed methods and subsequently draw conclusions based on that in Chapter 7.

[1] "ORTEC is one of the largest providers of advanced planning and optimization solutions and services. ORTEC's products and services result in optimized fleet routing and dispatch, vehicle and pallet loading, workforce scheduling, delivery forecasting, logistics network planning and warehouse control. ORTEC offers stand-alone, configurable and SAP® certified and embedded solutions, supported by strategic partnerships. ORTEC has over 1,800 customers worldwide, 700 employees and offices in Europe, North America, South America and the Pacific Region." `http://www.ortec.com/`

Chapter 2

Literature review

The vehicle routing problem (VRP) was first formulated by Dantzig and Ramser (1959), although they did not yet call it the VRP at that time. Since then, a tremendous amount of research has been devoted to this problem. Many extensions of the VRP have been described and for all of them exact and heuristic methods have been proposed. For a very extensive description of the single-depot VRP extensions and methods to solve them, both exact and by heuristics, we refer to Toth and Vigo (2001). For a list of literature devoted to the multi-depot VRP (MDVRP) and its extensions, see Salhi et al. (2013, Table 1). In the following sections we will discuss different enhanced variants of the VRP and how they differ from our problem of vehicle routing to deliver multiple products with multi-trip shifts and sourcing volume constraints at depots.

2.1 Multi-depot heterogeneous VRP with time windows

The multi-depot heterogeneous vehicle routing problem with time windows (MDHVRPTW) is a multi-depot VRP where distribution is done by a fleet with mixed characteristics. Next to that, customers have to be serviced within specified time windows. This realistic variant of the multi-depot VRP was first introduced in literature by Dondo and Cerdá (2007). They formulated the problem mathematically and proposed a cluster-based optimization heuristic. The heuristic is a three-phase approach, where in the first phase customers are clustered such that each cluster can be assigned to any vehicle without violating constraints. In the second phase clusters are assigned to vehicles and vehicles to depots by solving a mixed integer linear program (MILP). In the third phase the customer sequence within the clusters is optimised. Dondo and Cerdá (2007) solved some multi-depot benchmark problems with a running time between 30 and 60 minutes.

A branch-and-cut-and-price method for the MDHVRPTW has been developed by Bettinelli et al. (2011). They used this approach as an exact method and in a heuristic procedure. They only tested the heuristic on single-depot benchmark problems as they compared the heuristic with methods that have only been applied to single-depot problems.

Even more recently, Xu et al. (2012) developed a variable neighbourhood search (VNS) method to solve the MDHVRPTW. The VNS uses the best-improvement strategy for quick improvement of solutions but they also applied simulated annealing for diversification. Their heuristic is also only tested on single-depot benchmark problems. Xu and Jiang (2014) applied the same method as Xu et al. (2012) on a small multi-depot case with 16 customers.

Although the MDHVRPTW contains a lot of constraints similar to the ones we consider, there are also many restrictions that are not included. In the previously listed literature multi-trip shifts are not allowed, hence the requirement is that a truck visits a depot only at the beginning and end of a shift. For applications in the oil, gas and chemicals industry this is certainly not common as generally very few customers can be served with one truckload (in the fuels industry on average even less than two) and therefore a driver can do multiple trips per shift. Furthermore, in the MDHVRPTW we talk about the delivery of just one type of product, whereas most applications that we consider include multiple products. With this also comes the complication of compartments that can only carry one product.

2.2 Petrol station replenishment problem

The petrol station replenishment problem (PSRP) considers many constraints that are specific to the oil, gas and chemicals industry. Included is that trucks are composed of a number of compartments that can carry just one type of product each. The problems consider multiple products and allow multiple trips per shift. Furthermore, in many papers addressing the PSRP, the amount of a certain product to deliver to a certain customer is not fixed. Only a lower and upper bound on the amount to deliver is given, which means that the actual amount to deliver is an additional decision that has to be taken.

The PSRP was formulated for the first time by Cornillier et al. (2008). They considered the single-depot case with homogeneous vehicles and no time windows, so the focus was on the fact that only lower and upper bounds on the volumes to deliver are given. Cornillier et al. (2008) decomposed the problem into the tank truck loading problem, where they also took the decision of how much to deliver to each customer, and the routing problem. At about the same time, Ng et al. (2008) did a small-scale case study on petrol station replenishment in Hong Kong. Although they took multiple depots and a heterogeneous fleet into account, they assumed a fixed demand for every product and every customer. Their heuristic to assign customers to depots is based on geographic constraints and expert knowledge. For the assignment of customers to trips and trucks they solved a MILP. Cornillier et al. (2009) solved the heterogeneous fleet variant of the single-depot PSRP with time windows. They formulated the problem as a MILP and proposed to solve it by either removing infeasible arcs (pairs of customers that cannot be visited subsequently) or dividing the problem into geographical regions.

Surjandari et al. (2011) were the first to publish a mathematical formulation for the multi-depot variant of the PSRP, although they assumed that every customer has a fixed demand for every product. They also allowed multiple deliveries to the same customer. A tabu search heuristic was applied on a case with 208 customers, but details about the tabu search other than the parameter selection are not reported. Most recently, Cornillier et al. (2012) solved a multi-depot variant of the PSRP with heterogeneous vehicles and time windows where the amount to deliver to customers also has to be determined. The assumption of a fixed distribution of vehicles over the depots allowed them to apply a set covering approach. They proposed a heuristic that selects a subset of all routes and then solves the set covering problem. They apply this method to multi-depot problems with up to 30 customers.

Despite the large number of variants of the PSRP that have been addressed before, to the best of our knowledge there is no literature that also considers restrictions on the volume sourced from depots.

2.3 Assignment algorithms

As we are interested in splitting the problem into an assignment phase and a routing phase, we will now briefly discuss literature devoted to algorithms that assign customers to depots. Giosa et al. (2002) compared assignment algorithms that are used in a cluster first-route second environment. Although they solved the MDVRPTW with limited product availability at depots (maximum sourcing volume constraints), they did not mention how they incorporated the remaining volume available at depots in their assignment algorithms. The different heuristic algorithms that Giosa et al. (2002) applied are:

- Algorithms based on assignment through the urgency of a customer to be assigned to its closest depot (inspired by the insertion algorithm of Potvin and Rousseau (1993)).
- An assignment algorithm where for each depot the customer closest to the customer last assigned to the depot is also assigned to that depot.
- Two algorithms that do assignment by clusters, differing in the criteria on which the assignment of customers to clusters is based.

They compared the algorithms by applying a simple routing heuristic based on the savings algorithm (Clarke and Wright, 1964), and assumed that the clustering algorithm that yields the lowest total costs after routing is the best one. They show that the algorithms that assign by clusters give the best results.

Following on the article discussed above, Tansini and Viera (2006) proposed assignment algorithms that integrate time windows in the proximity measures used in the different assignment algorithms. They assumed that customers with similar time windows are favourable to be served by one vehicle and should therefore have an increased probability of being assigned to the same depot. After comparing the new assignment algorithms with some existing ones, Tansini and Viera (2006) concluded that incorporating time windows does yield better results, so it seems promising to include even more details about the problem in the clustering phase. Therefore, we will develop a heuristic that does this too.

Chapter 3

Problem definition

The routing network can be represented by a directed graph $G = (M \cup N \cup O, A)$ where M is the set of depots, N the set of clients that have demand for at least one product within the current scheduling horizon, and O is the set of start and end locations of vehicles. The set A contains the directed edges connecting node i to node j where $i \in M \cup N \cup O$, $j \in M \cup N \cup O$, $i \neq j$. We consider the distribution of a set of products P by a set of vehicles K. A vehicle is composed of a unique truck, trailer, and driver.

For every client $n \in N$ and product $p \in P$ there is a given demand d_{np}. Every customer has to be served during a single visit. To this end we assume that the demand of the requested products fit in one vehicle. Furthermore, the vehicle k serving customer n has to be compatible with the tools present at the customer site. For this reason we use binary parameter e^{vc}_{kn} that is 1 if vehicle k is indeed compatible with customer n and 0 otherwise (where the superscript 'vc' stands for vehicle-customer).

Every vehicle $k \in K$ has a set of compartments C_k. For every compartment $c \in C_k$ there is a given capacity l_c that cannot be exceeded. Products stored in a compartment can be used to serve multiple customers but different products cannot be stored in the same compartment. Vehicles should source all products of a trip at a single compatible depot, where compatibility between vehicle k and depot m is denoted by value 1 of binary parameter e^{vd}_{km} and 0 otherwise (where we use 'vd' to denote vehicle-depot). Every customer n should also be compatible with the depot m at which the assigned vehicle has loaded, which is true if binary parameter e^{dc}_{mn} is 1 and not true if it has value 0 ('dc' stands for depot-customer). In short, every customer $n \in N$ has to be served by a vehicle $k \in K$ which has loaded at a depot $m \in M$ such that $e^{\text{vd}}_{km} = e^{\text{vc}}_{kn} = e^{\text{dc}}_{mn} = 1$. Within the route scheduling horizon not less than q^{-}_{mp} or more than q^{+}_{mp} of product p can be sourced from depot m, for all $p \in P$, $m \in M$. To ensure feasibility we assume that the minimum and maximum amounts of available products at the depot are such that all demand can be satisfied.

Every vehicle $k \in K$ has a time window $[u_k, v_k]$ in which it can operate for a maximum of w_k hours. The working time of a customer will start with driving from the start location to the first depot and end with driving from the last customer to the end location (identical to the start location). A vehicle can do multiple trips within one shift, where a trip starts with one loading job followed by at least one unloading job. The trailer has to be completely empty at the end of a trip and every vehicle $k \in K$ should start and end at its location o_k. Service of customers $n \in N$ has to start within a given time window $[a_n, b_n]$ and service times at customer locations are composed of a fixed service time sf_n and a variable service time sv_n per unit-volume delivered. Loading times at depots $m \in M$ are in the same way composed of a fixed service time sf_m and a variable service time sv_m per unit-volume loaded. Of every edge $(i, j) \in A$ we know the distance g_{ij} and the driving time t_{ij}.

The goal is to minimise total costs subject to the mentioned constraints. The costs are composed of transport costs and supply costs. Transport costs of vehicle $k \in K$ are α_k per travelled kilometre and β_k per worked hour. Supply costs of product $p \in P$ at depot $m \in M$ are δ_{mp} per unit-volume supplied.

In some applications of this problem the demand for a product $p \in P$ of a customer $n \in N$ is not fixed but has to be between a certain minimum demand d^-_{np} and maximum demand d^+_{np}, such that $0 \leq d^-_{np} \leq d^+_{np}$. Supplying a value of at least the minimum is satisfactory, but it is beneficial to deliver a higher volume because the next delivery can then be further delayed. This makes the problem more difficult as the actual supplied volume to a customer becomes an additional decision that has to be made. In the main text of this thesis we will assume a fixed demand d_{np}, for all $n \in N$, $p \in P$. In Appendix A we present the untested ideas that we develop to deal with demand windows instead of a fixed demand.

Chapter 4

Cluster first-route second approach

In this chapter we introduce the assignment methods that we use in our cluster first-route second approach. In the traditional cluster first-route second approach every customer is assigned to a single depot. The reason for this is that in those cases the cluster first-route second approach is applied because the routing method cannot solve the multi-depot VRP. By assigning every customer to just one depot, the problem can be split up into $|M|$ single-depot VRP's.

In this thesis, on the other hand, we assume that the available routing method can solve multi-depot VRP's but not handle constraints on the sourced volume. Therefore, instead of assigning every customer to a single depot we can assign every customer to multiple depots, as long as no sourcing volume constraints are violated when a depot serves any subset of the customers that are assigned to it.

To illustrate the advantage of assigning customers to multiple depots, we make use of Figure 4.1. The example contains two depots (squares) and six customers (circles). For simplicity we consider the delivery of only one product. Next to the shown parameter values the minimum sourcing volumes are zero. When assigning every customer to just one depot, the optimal solution (if we take distance as expected costs of serving a customer from that depot) is to assign customers 1, 2, and 3 to depot 1 and customers 4, 5, and 6 to depot 2. The result is that still one unit-volume is available at depot 1 and two units at depot 2. If we allow for the assignment of customers to multiple depots we can give the routing heuristic more possibilities. In Figure 4.1 this is done by also assigning customer 5 to depot 1 and customer 3 to depot 2, as illustrated by the dashed 'territories'.

The increased freedom that the routing heuristic has can be of great value when factors that are not considered in the assignment phase turn out to be restricting. An example of this in Figure 4.1 would be that near depot 2 there are only vehicles with a capacity of 3 units. Now that customer 5 can also be

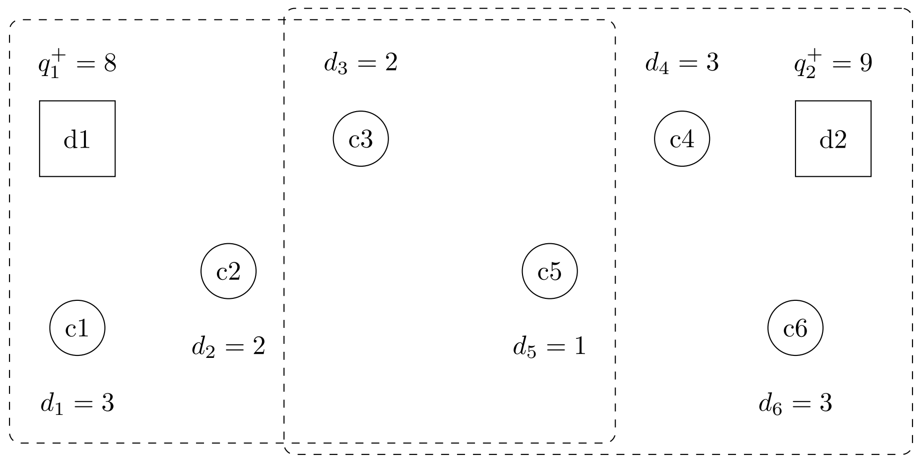

Figure 4.1: Example of advantage when allowing customers to be assigned to multiple depots. Minimum sourcing volumes q_1^- and q_2^- are both zero.

served from depot 1, there may be no need to have three trips from depot 2 but serve customer 5 on the trips from depot 1.

If we approximate a certain cost of serving a specific customer from a specific depot and minimise total costs of assigning every customer to depots, we will never assign a customer to multiple depots as that comes with increased costs. One of the ways to obtain an assignment to multiple depots is to maximise the total number of times customers are assigned to a depot (the number of depots to which a customer is assigned, summed over the customers). Focussing completely on the number of assignments is not beneficial though. This is illustrated by Figure 4.2, where one would normally assign customers 1 and 2 to depot 1 and customer 3 and 4 to depot 2 and leave one unit-volume of depot 2 unassigned. If we maximise the number of assignments we get the assignment shown in the example. In this case the total number of assignments has risen from four to five but this assignment will most likely lead to bad solutions as customer 3 is only assigned to depot 1.

To avoid situations as described in the previous paragraph, we split the assignment phase in two steps. In the first step we minimise total costs of assigning every customer to a single depot. Methods to do this are described in Section 4.1. The second step consists of expanding the solution by assigning every customer to other depots. The methods that we develop for that are discussed in Section 4.2.

4.1 Single-depot assignment stage

The classical assignment in a cluster first-route second approach results in the assignment to a single depot for every customer. In this section we first introduce

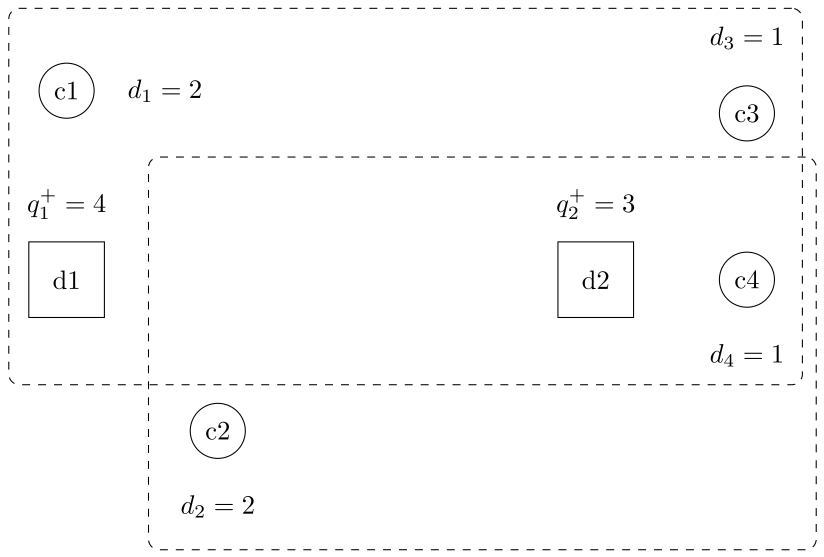

Figure 4.2: Example of when only maximising the number of assignments leads to strange optimal solutions. Minimum sourcing volumes q_1^- and q_2^- are both zero.

an exact method to do the assignment, by extending the well-known generalised assignment problem. Because the problem is NP-hard (Fisher et al., 1986) and the practical application requires short running times, we also develop a heuristic based on an urgency ordering of customers. However, the urgency heuristic only uses a small amount of the available operational data. Because we want to see if using more data increases the quality of the solutions we also describe a clustering heuristic that takes into account vehicle capacities, time windows, and customer service times.

Note that it is very well possible that for a problem instance the heuristics cannot find a feasible single-depot assignment whereas solving the exact formulation does yield a feasible result. As this has not happened in our experiments and is unlikely when using the methods in practice (because sourcing volume constraints will not be very strict for many depots) we have not developed methods to handle such a possible infeasibility.

4.1.1 Extended generalised assignment problem

The generalised assignment problem (Cattrysee and Van Wassenhove, 1992) is the problem of creating the least-cost assignment of a set of tasks to a set of agents who have limited resources available. Costs and resource requirements of having a certain agent perform a certain task are given for all agents and tasks.

We can simply adapt this problem to our assignment problem when we relax all constraints related to vehicles and time windows. We use γ_{mn} to denote the approximated cost of serving customer n from depot m, for all $m \in M$, $n \in N$. In the following formulation, the binary decision variable x_{mn} is 1 if customer n is being served from depot m and 0 otherwise, for all $m \in M$, $n \in N$.

$$\begin{aligned}
\min \quad & \sum_{m\in M}\sum_{n\in N}\gamma_{mn}x_{mn}, && && (4.1)\\
\text{s.t.} \quad & q^{-}_{mp} \le \sum_{n\in N} x_{mn}d_{np} \le q^{+}_{mp}, && \forall m \in M, p \in P, && (4.2)\\
& \sum_{m\in M} x_{mn} = 1, && \forall n \in N, && (4.3)\\
& x_{mn} \le e^{\text{dc}}_{mn}, && \forall m \in N, n \in N, && (4.4)\\
& x_{mn} \in \{0,1\}, && \forall m \in M, n \in N. && (4.5)
\end{aligned}$$

The objective function (4.1) that we minimise is the total approximated cost of serving customers from the assigned depots. Constraints (4.2) make sure that for every depot and product the supplied volume is between the given minimum and maximum when all customers are supplied the fixed demand volume. Restrictions (4.3) enforce that every customer is supplied from exactly one depot, and set (4.4) ensures that a customer is not assigned to a depot with which it is not compatible. Constraints (4.5) enforce the assignment variables to only take binary values.

It can easily be seen that our extended generalised assignment problem (GAP) with minimum sourcing constraints is NP-hard as the GAP is reducible to our problem by setting $q^{-}_{mp} = 0$ for all $m \in M$, $p \in P$, and the GAP is proven to be NP-hard (Fisher et al., 1986). Because we want to be sure that we can find solutions quickly we will discuss various heuristics after we propose how to approximate the costs of assigning a customer to a depot.

Approximated costs of serving a customer from a specific depot

The value of γ_{mn} can be decomposed into estimations of supply costs and transport costs. The supply costs are $\sum_{p\in P}\delta_{mp}d_{np}$.

To estimate the transport costs we first need to introduce some additional notation. We denote the average transport cost per kilometre by $\bar{\alpha} = \sum_{k\in K}\alpha_k/|K|$ and the average cost per worked hour by $\bar{\beta} = \sum_{k\in K}\beta_k/|K|$. For the average capacity of a vehicle we take $\bar{l}_{\text{tot}} = \sum_{k\in K}\sum_{c\in C_k} l_c/|K|$ and the total volume supplied to customer n is $d_{n,\text{tot}} = \sum_{p\in P} d_{np}$. In the following paragraphs we use ATU to denote the average truck utilisation of a truck. This is the fraction of the capacity of a vehicle that is used on average.

We propose to estimate the transport costs by summing the following components:

- Travelling costs to the customer and back to the depot: $2\bar{\alpha}g_{mn}\frac{d_{n,\text{tot}}}{ATU\cdot\bar{l}_{\text{tot}}}$.
- Costs for fixed loading time at the depot: $\bar{\beta}sf_m\frac{d_{n,\text{tot}}}{ATU\cdot\bar{l}_{\text{tot}}}$.

- Costs for variable loading time at the depot: $\bar{\beta} sv_m d_{n,\text{tot}}$.
- Costs for driving time to the customer and back to the depot: $2\bar{\beta} t_{mn} \frac{d_{n,\text{tot}}}{ATU \cdot \bar{l}_{\text{tot}}}$.
- Costs for fixed unloading time at the customer: $\bar{\beta} sf_n$.
- Costs for variable unloading time at the customer: $\bar{\beta} sv_n d_{n,\text{tot}}$.

For the costs of serving customer n from depot m this yields

$$\gamma_{mn} = \sum_{p \in P} \delta_{mp} d_{np} + \frac{d_{n,\text{tot}}}{ATU \cdot \bar{l}_{\text{tot}}} (2\bar{\alpha} g_{mn} + \bar{\beta} sf_m + 2\bar{\beta} t_{mn}) + \bar{\beta}(sf_n + d_{n,\text{tot}}(sv_m + sv_n)). \quad (4.6)$$

Note that only a share of the travelling costs, costs for fixed loading time, and costs for driving time are seen as costs for customer n. This is done as more than one customer may be served by the same vehicle and therefore those costs are split over several customers. The proportion accounted to customer n is the fraction that the demand of customer n will fill of an average vehicle with average truck utilisation. Thus, the full costs are passed on to a customer if the customer's total demand is exactly the capacity of an average vehicle with average truck utilisation.

We completely ignore the costs related to the vehicle driving from its start location to the first depot and from the end depot to the end location. The same holds for the costs made for travelling from customer to customer within a trip. By doing this we assume that those costs are only a small fraction of the total costs related to serving a particular customer.

The costs of serving a customer from a specific depot are well approximated by this approach when a customer is closely surrounded by many other customers. When customers are further apart and still served in the same trip, this approach will underestimate the costs. We hypothesise that this does not have a large influence on the assignment results as the underestimation for a specific customer will happen for all depots. Thus, the only effect of underestimating the assignment costs for all depots is that a difference in cost factors that are taken into account (e.g. different product prices at different depots) is less important than assumed. Therefore, the relative attractiveness of specific depots will lead to roughly the same result as long as the omitted cost factors are not a large fraction of the total costs.

4.1.2 Urgency-based heuristic

Our urgency-based assignment heuristic is inspired by the parallel assignment through urgencies of Giosa et al. (2002). The general idea is to measure an urgency for each customer. Urgency resembles the need to assign that customer based on the approximated costs of assigning customers to depots. There are two (competing) urgency measurements that we use.

The first urgency measure is the average cost of assigning a customer to a feasible depot minus the cost of assigning the customer to the cheapest feasible depot. Hence, the customer with the highest urgency has the highest need to be assigned to the cheapest feasible depot and is therefore subsequently assigned to it. This urgency measure fully focusses on assignment costs and does not take into account the minimum sourcing volume constraints.

The second urgency measure focusses on satisfying the minimum sourcing volume constraints as quickly as possible. It is calculated by taking the cost of assigning a customer to the cheapest feasible depot minus the cost of assigning the customer to the depot that currently delivers the least volume above the minimum sourcing volume of the product for which the customer has most demand. In mathematical terms, let f_{mp} denote the volume by which the minimum sourcing constraint of product p at depot m is exceeded. Initially $f_{mp} = -q^{-}_{mp}$ for all $m \in M$, $p \in P$. Then, the depot that sources the least volume above the minimum sourcing volume is $m_{\min} = \arg\min_{m \in M} f_{m,p^*}$, where $p^* = \max_{p \in P} d_{np}$ is the product that customer n has most demand for. The cheapest feasible depot is $m^* = \arg\min_{m \in M_n} \gamma_{mn}$. Therefore, the urgency calculated with the second measurement is $\mu_n = \gamma_{m^*,n} - \gamma_{m_{\min},n}$. As a result, all urgencies are smaller or equal to zero and the customer with the highest urgency will induce the smallest cost increase when assigned to the depot with the least volume above the minimum sourcing volume assigned. For that reason the customer is assigned to that depot ($m_{\min}$).

The algorithm of the urgency-based heuristic is shown in Algorithm 1. Until all customers are assigned to a depot (line 9), the urgency-based heuristic determines for every unassigned customer by which depots it can be served (lines 11–16) and what the urgency of that customer is (line 17). Subsequently, the customer with the highest urgency is assigned to a depot (which depot depends on the urgency measure) (lines 19–21).

After every customer is assigned, a procedure is started to ensure all minimum sourcing volume constraints are satisfied. A description of this procedure can be found in Algorithm 2. The procedure iteratively focuses on the depot m which has the 'least satisfied minimum sourcing constraint' (the value by which the minimum sourcing constraint is violated). Then, customers are ordered in non-increasing order of costs of assigning that customer to depot m. Starting with the cheapest customer, we check whether the customer n is compatible with the depot m, has demand for the product with least satisfied volume at depot m, whether no maximum sourcing volume constraints at depot m are violated if it also has to serve customer n, and if not serving the customer from the depot to which it was previously assigned does not violate any minimum sourcing volume constraints at that depot. If all checks are passed, the assignment of the customer is moved to depot m.

We hypothesise that the urgency-based heuristic works best when the maximum sourcing volume constraints are most restricting, because the procedure that makes sure minimum sourcing volume constraints are satisfied is relatively simple. The main simplification is made by switching the assignment of a customer if it has at least some demand for the product of which the minimum

Algorithm 1 Assignment through urgencies.

```
1: for all $m \in M$ do
2:     for all $p \in P$ do
3:         remaining capacity $r_{mp} = q^+_{mp}$
4:         capacity above minimum assigned $f_{mp} = -q^-_{mp}$
5:     end for
6: end for
7: assignment variable $x_{mn} = 0$ for all $m \in M$, $n \in N$
8: list of unassigned customers $N' = N$
9: while $N' \neq \emptyset$ do
10:     for all $n \in N'$ do
11:         list of depots to which customer $n$ can be assigned $M_n = \emptyset$
12:         for all $m \in M$ do
13:             if $e^{\mathrm{dc}}_{mn} == 1$ and $r_{mp} \geq d_{np}\ \ \forall p \in P$ then
14:                 $M_n = M_n \cup \{m\}$
15:             end if
16:         end for
17:         determine urgency $\mu_n$ based on chosen measurement
18:     end for
19:     most urgent customer $n^* = \arg\max_{n \in N'} \mu_n$
20:     determine depot $m^*$ to which $n^*$ will be assigned
21:     $x_{m^*,n^*} = 1$
22:     for all $p \in P$ do
23:         $r_{m^*,p} = r_{m^*,p} - d_{n^*,p}$
24:         $f_{m^*,p} = f_{m^*,p} + d_{n^*,p}$
25:     end for
26: end while
27: $(f, r, x) = \mathit{fixMinimumViolations}(f, r, x)$ (see Algorithm 2 on page 16)
```

sourcing constraint is not yet satisfied. Therefore, the procedure may reassign many customers with small demand for that product to satisfy the minimum sourcing volume constraints, whereas switching the assignment of one customer with large demand may be better.

The methods discussed up to now have not made use of operational data like time windows and vehicle information. This is probably not a problem when there is a very large number of customers ($|N| \to \infty$) because at every depot there will still be a sufficient variety of time windows and demand volumes to form 'efficient' trips. It becomes increasingly problematic though when there are fewer customers, because one risks assigning a set of customers to a depot just based on estimated costs, while those customers cannot be efficiently planned together. In the next section we describe a heuristic method that overcomes this disadvantage.

Algorithm 2 $(f, r, x) = \mathit{fixMinimumViolations}(f, r, x)$

```
1: least satisfied minimum sourcing constraint $lsmc = \min_{m\in M} \min_{p\in P} f_{mp}$
2: while $lsmc < 0$ do
3:    depot with least satisfied constraint $m_1 = \arg\min_{m\in M}(\min_{p\in P} f_{mp})$
4:    product that is least satisfied $p_1 = \arg\min_{p\in P} f_{m_1,p}$
5:    sort customers $n \in N$ by non-decreasing value of $\gamma_{m_1,n}$
6:    current customer $n = 1$
7:    boolean $\mathit{customerFound}$=false
8:    while $\mathit{customerFound}$==false and $n \le |N|$ do
9:       determine depot $m_2$ that currently serves customer $n$
10:      if ($e^{\text{dc}}_{m_1,n} == 1$ and $d_{n,p_1} > 0$ and
         $\nexists p \in P : r_{m_1,p} < d_{np}$ and $\nexists p \in P : f_{m_2,p} < d_{np}$) then
11:         $\mathit{customerFound}$=true
12:         $x_{m_1,n} = 1$
13:         $x_{m_2,n} = 0$
14:         for all $p \in P$ do
15:            $r_{m_1,p} = r_{m_1,p} - d_{np}$ and $r_{m_2,p} = r_{m_2,p} + d_{np}$
16:            $f_{m_1,p} = f_{m_1,p} + d_{np}$ and $f_{m_2,p} = f_{m_2,p} - d_{np}$
17:         end for
18:      end if
19:      $n = n + 1$
20:   end while
21:   if $\mathit{customerFound}$=false then
22:      no feasible solution found, terminate heuristic
23:   end if
24:   least satisfied minimum sourcing constraint $lsmc = \min_{m\in M} \min_{p\in P} f_{mp}$
25: end while
```

4.1.3 Clustering customers

To include an increased number of operational data in the assignment process we create clusters of customers. The idea is to cluster customers who are favourable to be served in the same trip. The advantage of doing so is that any type of constraint can be taken into account when deciding whether to cluster certain subsets of customers or not. Examples of these constraints are time window constraints and compatibility with depots and vehicles.

After clusters of customers are created they can be assigned to depots by means of a slightly adapted version of the extended GAP described in Section 4.1.1. We hypothesise that the exact assignment algorithm will always terminate within reasonable time when using clusters, because assigning clusters of customers decreases the size of the problem. We denote the set of clusters by Y, the set of customers who are part of cluster y by N_y, and the binary decision variable that is 1 if cluster y served by depot m and 0 otherwise by x^{cluster}_{my}.

Equation (4.3) then has to be replaced by

$$\sum_{m \in M} x_{my}^{\text{cluster}} = 1, \quad \forall y \in Y, \tag{4.7}$$

and we have to add

$$x_{mn} \geq x_{my}^{\text{cluster}}, \quad \forall y \in Y, n \in N_y. \tag{4.8}$$

The two above sets of constraints ensure that every cluster is served by exactly one depot, and that serving a cluster from a certain depot implies that every customer in that cluster has to be served from that depot. In the latter set an inequality is enough because x_{mn} will always be chosen as low as possible since costs are minimised.

We hypothesise that introducing the additional constraints is recognized by the MILP solver as a decrease of the problem size. The alternative would be to accumulate parameter values of customers in a cluster and use the original formulation of Section 4.1.1 to assign clusters to depots.

The general clustering algorithm is described in Algorithm 3. Details like the stopping criterion, selection method, and merging requirements are discussed in the next paragraph. The clustering procedure starts with a separate cluster for every customer. In every iteration two clusters are selected based on a certain selection criterion. Next, the clusters are either merged or not, based on one or multiple acceptance criteria. If the clusters are merged, cluster characteristics that are important in the selection process have to be recalculated. If two clusters are not merged, the pair will be omitted from the selection procedure from that point onwards. One of the two clusters can of course be selected for merging again, but only if it may be merged with a cluster with which it has not yet been considered for merging. The clustering will stop when a stopping criterion has been reached, or if no pairs of clusters satisfy the merging criteria.

We select two clusters based on the inter-cluster distance. The two closest clusters are always considered first. The philosophy behind this strategy is that it is most interesting to assign a number of nearby customers to the same depot because they are generally most attractive to be served on the same trip. Two selected clusters are merged into one cluster if they satisfy the following three criteria:

1. There is at least one depot that is compatible with every customer and has enough volume available of the products required by the customers.

2. There is at least one vehicle that is compatible with every customer, is compatible with at least one of the depots that satisfies the requirement mentioned above, and has enough compartments and compartment capacity to carry the demand volumes of all customers.

3. Customer time windows, service times, and driving times allow all customers to be serviced within one trip.

Algorithm 3 General clustering algorithm.

```
1: start with every customer in a separate trip cluster $y_n$, $1 \leq n \leq |N|$
2: create list of cluster pairs $Y^p$
3: sort list $Y^p$ by non-decreasing inter-cluster distance
4: index $i = 1$
5: while stopping criterion not reached and $Y^p \neq \emptyset$ do
6:   select top cluster pair in $Y^p$, denoted by $(y_a, y_b)$ for some $a, b > 0$, $a \neq b$
7:   if $(y_a, y_b)$ satisfies merging requirements then
8:     merge $y_a$ and $y_b$ into one cluster $y_{|N|+i}$
9:     remove all entries related to $y_a$ or $y_b$ from $Y^p$
10:    calculate characteristics of cluster $y_{|N|+i}$
11:    calculate inter-cluster distances to $y_{|N|+i}$
12:    add all cluster pairs related to $y_{|N|+i}$ to $Y^p$
13:    re-sort $Y^p$
14:    $i = i + 1$
15:  else
16:    remove entry related to the pair $(y_a,y_b)$ from $Y^p$
17:  end if
18: end while
```

Note that the third requirement actually involves solving a travelling salesman problem with time windows, which is a well known NP-hard problem. While this may be time consuming when merging large clusters we do not expect this to be a problem as the second criterion is strict enough to limit cluster sizes. Furthermore, if running times increase because of the third requirement we may decide to only test a limited number of permutations and reject the merge if none of those allow all customers to be serviced within one trip. For example, in the tests the run in Chapter 6 we have set a maximum of 1000 permutations, which means that for six customers we can still check all permutations.

If two selected clusters are merged, we have to update the inter-cluster distance to every other cluster. To this end, we simply take the average distance of any pair of customers where one customer is part of the first cluster and the other customer is part of the second cluster. This technique is known as group average and is a compromise between single link clustering (taking the minimum distance) and complete link clustering (taking the maximum distance). The advantage of group average clustering is that it is less sensitive to outliers than single link clustering (caused by what is called the chaining phenomenon) and does not break large clusters as much as complete link clustering.

We do not specify a stopping criterion other than continuing until no more cluster pairs satisfy the merging criteria, as our merging criteria are very strict. Especially the second condition will ensure that running times will not be large if we let the clustering algorithm run until no pair of clusters satisfies the requirements.

4.2 Multi-depot assignment stage

As our routing heuristic can solve the multi-depot VRP, we can also assign customers to multiple depots as long as no sourcing volume constraints are violated if a depot serves any subset of the customers assigned to it. We first describe a mathematical formulation where we maximise the total utility of assigning more depots to customers. Then we also introduce a greedy heuristic that guarantees short running times.

4.2.1 Extended multi-dimensional knapsack problem

The multi-dimensional knapsack problem (MKP) is the problem of maximising the utility of selected items such that the capacity constraints of a number of resources are respected. Every item has a given utility and takes up a given amount of every resource considered. For a detailed overview of the problem and proposed methods to solve it, see Fréville (2004). We can extend the idea of the MKP to maximise utility of assigning additional customers to every depot with the remaining capacity.

The utility θ_{mn} of assigning a customer $n \in N$ to a depot $m \in M$ will be specified later. We also make use of the additional decision variable z_{mn} which can only be 1 if customer n is not assigned to any depot other than depot m and has to be 0 otherwise, for all $m \in M$, $n \in N$.

$$
\begin{aligned}
\max \quad & \sum_{m\in M}\sum_{n\in N} \theta_{mn} x_{mn}, && && (4.9)\\
\text{s.t.} \quad & \sum_{n\in N} d_{np} x_{mn} \le q^{+}_{mp}, && \forall m \in M, p \in P, && (4.10)\\
& z_{m_1 n} \le 1 - x_{m_2 n}, && \forall m_1 \in M, m_2 \in M \setminus \{m_1\}, n \in N, && (4.11)\\
& \sum_{n\in N} d_{np} z_{mn} \ge q^{-}_{mp}, && \forall m \in M, p \in P, && (4.12)\\
& x_{mn} = 1, && \forall m \in M, n \in N_m, && (4.13)\\
& x_{mn} \le e^{\text{dc}}_{mn}, && \forall m \in M, n \in N \setminus N_m, && (4.14)\\
& x_{mn} \in \{0,1\}, && \forall m \in M, n \in N \setminus N_m, && (4.15)\\
& z_{mn} \in \{0,1\}, && \forall m \in M, n \in N. && (4.16)
\end{aligned}
$$

The objective function (4.9) is the total utility of made assignments. With constraint set (4.10) we make sure that the maximum capacity of the depot is not exceeded. Restrictions (4.11) enforce z_{mn} to be zero if there is at least one depot other than m to which customer n is also assigned. Because of the way we define z_{mn} we can use constraints (4.12) to ensure that at least q^{-}_{mp} of a product $p \in P$ is sourced from a depot $m \in M$.

The set $N_m \subseteq N$ contains all customers that are assigned to depot m in the first phase, and therefore have to stay assigned in the second phase. This is ensured by restrictions (4.13). All other customers can be assigned if the depot is compatible, mathematically represented by set (4.14), and their assignment variable has to be zero or one, as stated by (4.15). Constraints (4.16) ensure binary values for all z_{mn} variables.

For the utility of assigning a customer $n \in N$ to a depot $m \in M$ we use $\theta_{mn} = \gamma_{m_1 n}/\gamma_{mn}$, where m_1 is the depot to which customer n was assigned in the single-depot assignment step. We use this value because it is most interesting to assign a customer n to depot m if the assignment costs are low and if the assignment costs of customer n to depot m_1 are high. Therefore, the utility function has to be non-increasing in γ_{mn} and non-decreasing in $\gamma_{m_1 n}$. The utility function also has to be strictly positive to ensure that there is always an incentive to add assignments.

The regular MKP can be solved with our formulation by setting $|M| = 1$ and $q^-_{mp} = 0$ for all $p \in P$. Because the MKP is NP-hard, as briefly described in Fréville (2004, Section 3.1), our problem is NP-hard too.

Limitations on the number of assigned depots

In some routing software packages there may be a limit on the number of depots to which a customer can be assigned if the customer is not assigned to all depots. Such constraints are easily added to the optimisation problem presented above. If a customer either has to be assigned to all depots or to at most L depots, we add constraints (4.17)–(4.19) to (4.9)–(4.16).

$$h_n|M| \leq \sum_{m\in M} x_{mn}, \quad \forall n \in N, \tag{4.17}$$

$$\sum_{m\in M} x_{mn} \leq L + h_n(|M| - L), \quad \forall n \in N, \tag{4.18}$$

$$h_n \in \{0, 1\}, \quad \forall n \in N. \tag{4.19}$$

Restriction set (4.17) ensures that customer n is assigned to all depots if binary decision variable h_n is 1, for all $n \in N$. Furthermore, constraints (4.18) make sure that customer n is assigned to at most L depots if binary decision variable h_n is 0, for all $n \in N$. Because h_n has to be either 0 or 1, as defined in (4.19), every customer $n \in N$ will be assigned to at most L depots or to all depots. Note that $|M|$ does not stand for the notorious 'Big M' but the number of depots.

Introducing the above set of constraints in the extended MKP will not only decrease the number of depots to which a customer is assigned from at most $|M| - 1$ to L, for customers who are not assigned to all depots. It may also increase the number of customers who are assigned to all depots as assigning a customer to more than L and less than $|M|$ depots is not allowed and therefore the probability that there is enough capacity to assign a customer to all depots is higher.

In addition, we propose to change (4.10) to only hold for all products $p \in P_m$ where P_m is the set of products that depot m offers. This is useful when for example all depots offer three products except for one depot that offers only two products. If there are no sourcing volume constraints and the assignment methods were run, that would result in all customers with demand for the product not available at the latter depot to be assigned to every depot but that one, while we can also assign the customer to that depot as the routing heuristic will never choose to serve the customer from that depot anyway. If limitations on the number of assigned depots would apply, the number of assigned depots would even further decrease. By setting (4.10) only for the products offered by a depot, this problem is solved and every customer will be assigned to every depot if no sourcing volume constraints are considered.

4.2.2 Greedy heuristic

The heuristic that we use in case the extended MKP cannot be solved within acceptable time is a greedy heuristic based on the utility defined in the previous section. The heuristic is described in Algorithm 4.

We sort customer-depot pairs that are not yet assigned based on the utility (non-increasing) (lines 9–10). Then we go through the list of pairs and check for every pair whether the customer n could also be assigned to the considered depot m without violating any constraints (lines 11–28). To this end we first assert that the depot is compatible with the customer and no maximum sourcing volume constraints are violated if depot m also serves customer n (lines 13–16). After that, we also check if the customer is currently only assigned to one depot (line 17). If the latter is true, we ensure that all minimum sourcing volume constraints are still satisfied if customer n is not served from that depot (lines 18–21). When there is no mentioned test that fails, it is possible to assign customer n to depot m too and the assignment is made (lines 23–27 and 30–34).

Limitations on the number of assigned depots

If the number of assigned depots has to be limited by L for customers who are not assigned to all depots, this is no problem with the greedy heuristic. After the greedy heuristic terminates one can identify which customers are assigned to more than L but not all depots. By keeping track of the order in which the depots are assigned to the customer, we remove the last added depots such that exactly L remain. The removed depots will not have a higher assignment utility than the depots that remain assigned to the customer.

It is important to note that we cannot simply stop assigning a customer to depots once it is already assigned to L depots. After all, at that point it is still possible that a customer will eventually be assigned to every depot and this is also satisfactory.

To ensure all customers can be served from any depot if no sourcing volume constraints are considered, we can change P in line 14 to P_m where again P_m is the set of products that depot m offers.

Algorithm 4 Heuristic to assign customers to multiple depots.

```
1: list of customers who are assigned to a single depot $N_1 = N$
2: for all $m \in M$ do
3:    list of customers $n$ depot $m$ does not serve $N_{m,-} = \{n | x_{mn} = 0\}$
4:    for all $p \in P$ do
5:       remaining capacity $r_{mp} = q^+_{mp} - \sum_{n \in N} x_{mn} d_{np}$
6:       capacity above minimum assigned $f_{mp} = \sum_{n \in N} x_{mn} d_{np} - q^-_{mp}$
7:    end for
8: end for
9: create list $A$ of customer-depot pairs $(m, n)$ for which $x_{mn} = 0$
10: sort list by non-increasing values of $\theta_{mn}$
11: for every entry in $A$ do
12:    current depot is $m$ and current customer is $n$
13:    boolean that depot $m$ can serve customer $n$ is $canServe = (e^{\text{dc}}_{mn} == 1)$
14:    if $\exists p \in P : r_{mp} < d_{np}$ then
15:       $canServe$=false
16:    end if
17:    if $canServe$==true and $n \in N_1$ then
18:       determine $m_1$, the only depot for which $x_{m_1 n} = 1$
19:       if $\exists p \in P : f_{m_1 p} < d_{np}$ then
20:          $canServe$=false
21:       end if
22:       if $canServe$==true then
23:          $N_1 = N_1 \setminus \{n\}$
24:          for all $p \in P$ do
25:             $f_{m_1 p} = f_{m_1 p} - d_{np}$
26:          end for
27:       end if
28:    end if
29:    if $canServe$==true then
30:       $x_{mn} = 1$
31:       for all $p \in P$ do
32:          $r_{mp} = r_{mp} - d_{np}$
33:       end for
34:    end if
35: end for
```

Chapter 5

Variable neighbourhood search heuristics

To be able to judge how competitive our developed cluster first-route second approach is, we develop variable neighbourhood search (VNS) heuristics. The idea of a VNS heuristic was first proposed by Mladenović and Hansen (1997) to overcome issues of traditional local search methods. The latter methods explore only one neighbourhood (i.e. solutions that differ in the same way from the previous solution) and therefore cannot escape from local optima found by searching in that neighbourhood. Contrary to those methods, VNS searches through multiple neighbourhoods which are ordered such that they contain increasingly different solutions.

The neighbourhoods that are most commonly explored for solving vehicle routing problems can be found through the k-opt exchange operators (first mentioned in the form of 2-opt by Croes (1958)). A k-opt exchange is the replacement of k arcs (links indicating that two specific locations are visited subsequently) by k different ones with the requirement that no disjoint sub-tours are formed. This approach is most often applied with $k = 2$ or $k = 3$, because higher values of k require much computation time while they do not yield many further improvements.

For $k = 2$ the move can be seen as inverting the order of a consecutive sequence of customers (Lin, 1965, Theorem 2) as shown in Figure 5.1. The orientation of the trip is preserved except for the sequence of locations b and c. In that sequence customers are served exactly the other way around than before. Note that in a 2-opt move removing two edges that have one endpoint in common will inevitably result in the same solution and therefore those moves are not considered. An example of a 2-opt exchange can be found in Figure 5.2. The trip goes from location 0 (the depot) to locations 1, 4, 3, 2, 5, and back to location 0. After the 2-opt exchange has been made the trip is $0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 0$. The order of visits of locations 4, 3, 2 has been inverted. Considering that $|N|$ customers have to be planned there are $\mathcal{O}(|N|^2)$ 2-opt

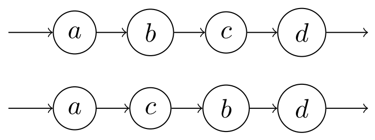

Figure 5.1: Clarification of how we define the inversion of the order of a consecutive sequence of customers.

moves for a given solution.

Furthermore, the 3-opt exchange operator is equivalent to relocating a consecutive sequence of customers and either inverting the order of customer visits of the sequence or not (Lin, 1965, Theorem 6). Individual customers are relocated by applying a 3-opt exchange where exactly two edges share an endpoint with another removed edge. Again, moves where every removed edge shares an endpoint with another removed edge are not useful 3-opt moves, because they result in either the same solution or a 2-opt move. A 3-opt move can also be applied to two trips, resulting in the relocation of a sequence of customers to a different trip (possibly in inverted order). For any solution there are $\mathcal{O}(|N|^3)$ 3-opt moves.

When customer time windows are very tight it may not be useful to search the complete neighbourhood of 3-opt exchanged solutions because inversion of the order of customer visits will cause time window violations. In such cases, it may be wise to use the Or-opt exchange operator (Or, 1976). This operator also relocates a sequence of customers, but does not invert the sequence. Therefore, the Or-opt neighbourhood is approximately half as large as the 3-opt neighbourhood (not exactly, as sequences of only one customer cannot be inverted).

Figure 5.3 shows an example of a 3-opt move. The original trip is $0 \to 1 \to 4 \to 5 \to 2 \to 3 \to 0$. After the 3-opt move is made the trip is $0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 0$. The sequence of nodes 4 and 5 is relocated from between nodes 1 and 2 to between nodes 3 and 0. This is also an example of an Or-opt exchange of length two, because in this particular example the order in which nodes 4 and 5 are visited is maintained.

Although 4-opt moves applied to two trips are not used a lot, some of them are very interesting because those involve the exchange of two sequences of customers. The latter moves are actually the same as what is called the CROSS exchange operator, proposed by Taillard et al. (1997). The order in which customers contained in the sequences are visited stays the same and helps keep the size of the neighbourhood limited. Figure 5.4 gives an example of a CROSS exchange. Originally, the first trip is $0 \to 1 \to 2 \to 3 \to 6 \to 0$ and the second trip $5 \to 4 \to 7 \to 8 \to 9 \to 5$. After the CROSS exchange the first trip is $0 \to 1 \to 2 \to 3 \to 4 \to 0$ and the second trip is $5 \to 6 \to 7 \to 8 \to 9 \to 0$. If the exchanged sequences can be of different lengths there are $\mathcal{O}(|N|^4)$ moves possible for any solution. When only sequences of equal length are exchanged the number of moves is reduced to $\mathcal{O}(|N|^3)$.

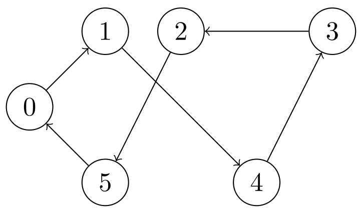
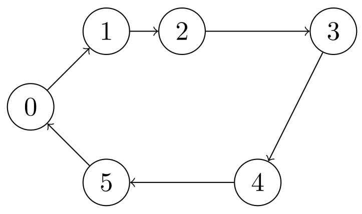
Figure 5.2: Example of a 2-opt exchange.

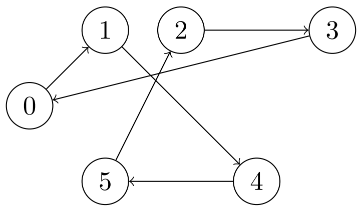
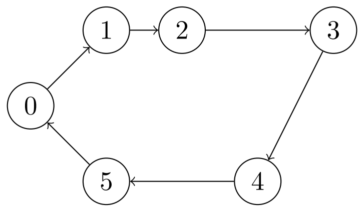
Figure 5.3: Example of a 3-opt and Or-opt exchange resulting in the relocation of a sequence of length two.

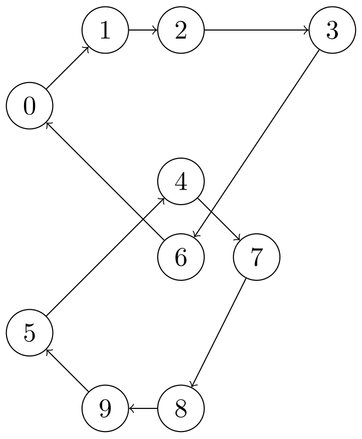
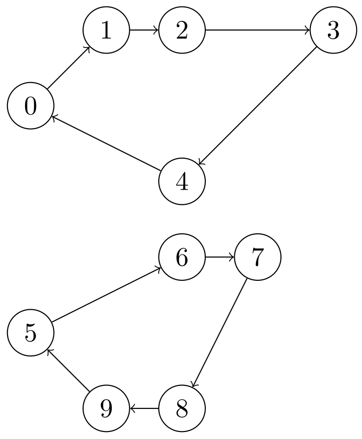
Figure 5.4: Example of a CROSS exchange of a sequence of length one.

We apply two types of variable neighbourhood search approaches. The first type is a repetitive version of a variable neighbourhood descent (VND), on which more can be read in Hansen and Mladenović (2001), and a destruct and repair procedure. The outline of this algorithm can be found in Algorithm 5 (page 27). In every iteration of the optimisation process we move through the neighbourhoods until an improvement is found. If one is found, we return to the first neighbourhood and repeat the same process. The iteration finishes when all neighbourhoods have been explored without finding an improvement. The second type is a regular variable neighbourhood search with the same destruct and repair procedure. The outline of that algorithm is described in Algorithm 6. The difference between a VND and a classic VNS is that a VND stays in a neighbourhood until all neighbours are evaluated or an improvement has been found, whereas a classic VNS only evaluates one random neighbour in every neighbourhood.

In the VND we check whether a solution-improving neighbour exists by iteratively going over all neighbours in the neighbourhood and calculating the costs of the solution if we move to that neighbour. In variable neighbourhood descent one traditionally moves to the best neighbour in the neighbourhood (best improvement) but because it can be very time consuming to calculate the costs of all solutions before making a choice we also generate results by moving to the first neighbour that decreases the costs (first improvement).

For how many iterations we let this algorithm run depends mostly on the running times. Therefore, we experiment with different values and choose a number of iterations that we think gives a good trade-off between speed and solution quality.

We describe our variable neighbourhood search approaches in four sections. First we describe how the initial solution is created. Then, we discuss the neighbourhoods that we explore. After that, we propose the destruct and repair method. Finally, we describe how we decide whether we accept a solution or not. In the section that discusses the neighbourhoods we will sometimes refer to a solution being better or worse than another. How we actually define that, made more complicated by the fact that we allow infeasible solutions, is described in Section 5.4. In short, when both solutions are feasible we compare by costs. When only one of a pair of solutions is feasible, it is better than the infeasible solution. When both solutions are infeasible we look at the value by which certain constraints are violated.

5.1 Building initial solution

When building the initial solution we aim to find a solution structure that has a high probability of being feasible or becoming feasible within a low number of moves to neighbouring solutions. To this end we follow the steps described below.

Algorithm 5 Global outline of our VND

```
1: build initial best solution $X_{\text{best}}$
2: make copy $X' = X_{\text{best}}$
3: while stopping criterion not met do
4:   set current neighbourhood $\kappa = 1$
5:   while $\kappa \leq \kappa_{\max}$ do
6:     if $\exists$ neighbour in $\kappa$-th neighbourhood of $X'$ that yields cost decrease
       then
7:       move to that neighbour with $X'$
8:       $\kappa = 1$
9:     else
10:      $\kappa = \kappa + 1$
11:    end if
12:  end while
13:  if costs of $X'$ are lower than of $X_{\text{best}}$ then
14:    update best solution $X_{\text{best}} = X'$
15:  end if
16:  destruct and repair $X_{\text{best}}$ and save it to $X'$
17: end while
```

Algorithm 6 Global outline of our VNS

```
1: build initial best solution $X_{\text{best}}$
2: make copy $X = X_{\text{best}}$
3: while stopping criterion not met do
4:   set $\kappa = 1$
5:   while $\kappa \leq \kappa_{\max}$ do
6:     generate random solution $X'$ from $\kappa$-th neighbourhood of $X$
7:     apply local search to find locally optimal solution $X''$ from $X'$
8:     if costs of solution $X''$ are lower than of solution $X$ then
9:       update best solution in current iteration $X = X''$
10:      $\kappa = 1$
11:    else
12:      $\kappa = \kappa + 1$
13:    end if
14:  end while
15:  if destruct and repair criterion applies then
16:    destruct and repair $X_{\text{best}}$ and save it to $X$
17:  end if
18: end while
```

Step 1: Assignment of customers to depots

To assign customers to depots we make use of the urgency-based heuristic with urgency measure 1 described in Section 4.1.2, because that method ensures that the constraints on the sourced volume are satisfied and has a short running time for real-world problem sizes.

Step 2: Assignment of vehicles to depots

We cannot easily judge what vehicle should serve which customers, so we let every vehicle start at a compatible depot while trying to keep the distance between the start location and the assigned depot short. First, to ensure that at every used depot a compatible vehicle starts, we iteratively assign to a used depot the closest (in terms of distance) compatible unassigned vehicle. This is done in non-decreasing order of the number of compatible unassigned vehicles. By a used depot we mean a depot to which at least one customer is assigned in the assignment procedure. To assign the remaining vehicles, we assign every vehicle one-by-one to the closest compatible used depot. For assignment to a depot, we arbitrarily order the vehicles lexicographically.

Step 3: Ordering of customers

We order customers by non-decreasing values of start of the serving time window a_n (for customer n). Customers with equal values of a_n are sorted by non-decreasing values of end of the serving time window b_n. For customers that also have equal values of b_n we break ties by total demand value $\sum_{p \in P} d_{np}$ where customers with a lower total demand should be listed higher. By assigning customers in this order we try to be able to solve time window violations with a small amount of neighbourhood moves. In case customers break ties for all the given ordering rules we arbitrarily order them lexicographically by name.

Step 4: Assignment of customers to vehicles

Until all customers are assigned to a trip, we select the next customer (from the list as ordered in Step 3) and assign it to a vehicle as follows. For all compatible vehicles assigned to the same depot as the customer, we calculate the total volume that it already has to deliver and divide this by the total capacity of the vehicle. This metric represents approximately how many trips the vehicle already has to drive. By choosing for this approach we do not keep track of the time windows and maximum time the vehicle can operate, but in the exploration of neighbourhoods we partly focus on solving such potential violations. If there is no compatible vehicle that is assigned to the same depot, we assign customers to an incompatible vehicle using the same metric. Again, in the exploration of neighbourhoods such violations may be solved quickly.

Step 5: Creation of trips

We add the depot visits to the vehicles to form trips. A sequence of customers is served in the same trip as long as the vehicle can carry all requested products. To this end we add customers to the trip under consideration until adding another customer violates the capacity constraints of the vehicle. From that customer onwards the next trip starts and the same process is repeated. This is equivalent to minimising the number of depot visits, but is not necessarily optimal in terms of travel time and distance.

5.2 Description of neighbourhoods

In every iteration of the VNS we explore a number of neighbourhoods. For a solution, each neighbourhood contains a set of solutions that can be found by applying the same type of change to the solution. The ordering of neighbourhoods is important because we explore the first neighbourhood each time an improvement is found and the last neighbourhood only if no improvements could be found in the other neighbourhoods. Therefore, the chosen neighbourhoods are ordered by increasing size and improvement they could make to the solution. We use the neighbourhoods described in the list below. When iterating over trips of a vehicle we always do this in the order of execution by the vehicle. We arbitrarily order the vehicles lexicographically.

Note that the ordering of neighbours within a neighbourhood can be important in the variants of our heuristic that apply first improvement. Therefore we explicitly describe the order for every neighbourhood. For the variants that use best improvement the ordering of neighbours within a neighbourhood is not important because all neighbours will always be evaluated before potentially making a move to one of those neighbours.

1. Changing the depot visit of a trip into another depot. For every depot visit that is selected we try to change it into the visit of any other depot (sorted lexicographically as this is arbitrary).

2. Interchanging the depot visited in one trip with the depot visited in another trip (also across vehicles). First we check for the interchange of the first depot visit of the first vehicle with every other depot visit of all vehicles. Then we do the same for the second depot visit of the first vehicle (or the first depot visit of the second vehicle if the first vehicle only visits a depot once), and so on.

3. Making an Or-opt exchange within the trip. We first try to relocate single customers to different places in the trip, starting with the first customer. New locations for the sequences are always tried from the front of the trip to the end of the trip. Then we continue with consecutive sequences of two customers, and so on, until the relocation of sequences that contain all but one customer of the trip.

4. Making an Or-opt exchange covering different trips of the same vehicle. Again, we first try to relocate single customers to different locations of other trips and if every single customer of a trip has been tried, we check for moves containing two customers. Only after all moves from a certain trip have been tried we try to relocate customers of the next trip. Note that we also consider relocation of complete trips. When a complete trip is relocated to a place in a different trip the depot visit of the original trip disappears.

5. Making an Or-opt exchange covering trips of different vehicles. Again, we first try to move single customers and only after all moves with a single customer have been tried we increase the length of the sequences moved. The sequences are moved from a given trip to all other locations of all trips of different vehicles before moving sequences from the second trip. The order of moves with the same sequence size is the same as in the first and fourth neighbourhood.

6. Making a CROSS exchange covering trips of different vehicles. The order of moves is the same as in the previous neighbourhood but now we swap two sequences of customers instead of moving a sequence from one place to another. We only exchange sequences of equal length to limit the size of this neighbourhood and speed up the heuristic.

7. Moving a trip to a different place in the same shift. For every vehicle we try to relocate trips to other places in the same trip. After all places have been tried, the next trip is selected for moving. This is done for all trips of a vehicle before moving trips of the next vehicle.

8. Moving a trip to another vehicle. Trips are ordered based on the vehicle that executes them and the order in which the trips are done. All places are tried for a trip before selecting the next trip to be relocated.

9. Swapping two trips (also of different vehicles). We first try to swap the first trip with any other trip and then move to the second trip.

10. Swapping two shifts. The vehicle pairs to swap shifts of are ordered by the first vehicle, breaking ties by the second vehicles.

The reason for this ordering is that the first and second neighbourhood are very small (when T is the set of trips they have $\mathcal{O}(|T||M|)$ and $\mathcal{O}(|T|^2)$ neighbours respectively). Changing the visited depot for a trip can prove to be very useful especially because in the initial solution a vehicle visits the same depot for all trips. The third neighbourhood is larger but still concerns neighbours that only differ by the sequence of customers within one trip.

From the fourth neighbourhood onwards we consider relocating or exchanging customer sequences between trips and vehicles. Note that when we move sequences of customers to other trips by means of an Or-opt exchange we also allow the relocation of the sequence into a new trip at the end of the shift. The reason for doing this is that moving sequences into existing trips will often result in an infeasible trip in terms of vehicle capacity. When a sequence is moved to the new trip at the end of the shift, the visited depot of that trip will be the depot to which the first customer of that trip was assigned in Step 1 of the initial solution build (see Section 5.1).

We do not invert the order of relocated sequences to limit the sizes of the neighbourhoods and exclude many moves that are unlikely to yield cost improvements (due to customer time window limitations). Furthermore, the sizes of most neighbourhoods heavily depend on the size that trips can generally have. In case the running times are too large it is possible to decrease the running time by limiting the size of the relocated sequences in the different neighbourhoods.

Although in general we are sure that the ordering of the neighbourhoods is correct, there are two things that we will test in the experiments of Chapter 6. Firstly, we check whether it may be better to optimise the order of customers within trips before changing the depot visit of trips (in terms of the current neighbourhood numbers that means that the neighbourhoods are ordered $3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$). Secondly, we evaluate whether we get better results when we swap two neighbourhoods if the first and the second only differ in the fact that the first contains neighbours where part of a trip or shift has been moved and the second contains neighbours where the same part of a trip or shift has been swapped. This holds for neighbourhoods 5 and 6, and for 8 and 9. We consider swapping to be a bigger change to the solution, but neighbourhoods with neighbours where something is swapped are approximately half the size of the neighbourhoods with neighbours where something is moved. To test what is best we use the neighbourhood ordering $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 8 \rightarrow 10$.

When moving to a neighbour we recalculate costs and violations of the schedule. The start and end times of a trip and the arrival times at customers are calculated from the end of the shift to the start. For the last customer we assume that service starts as early as possible. Then we go backwards through the shift and if service of a customer has to start before the start of the time window we delay all later visits (if needed). This means that in any solution we will never see that service of customers starts too early. Finally, if the calculated start of the shift is before the start of the operating window of the vehicle we delay the complete schedule as much as needed or possible without violating customer time window constraints.

In the classic variable neighbourhood search variants a local search procedure is run before a neighbour is compared to the best known solution. This is done because otherwise it is very unlikely that the neighbour is better than the best known solution. The local search that we apply is the third neighbourhood shown above. We choose this neighbourhood because it optimises most locally (within trips). For this reason the classic VNS has 9 neighbourhoods. When applying the local search procedure by a local search within neighbourhood 3, we both evaluate first improvement and best improvement variants.

5.3 Destruct and repair procedure

Once an iteration of the variable neighbourhood search terminates we have a locally optimal solution, but that solution may not be the global optimum. For that reason we destruct part of the solution and rebuild it before moving to the next iteration of the VNS. We apply this destruct and repair procedure to the currently best-known solution and not to the solution that came out of the current iteration, because the destruct and repair run of the previous iteration may have caused the solution to shift to a region with only solutions of very bad quality. The disadvantage of using the best solution is that we cannot use a fully deterministic destruct and repair procedure as that may cause our heuristic to cycle.

To improve the schedule we want to relocate all trips of a vehicle that adds a large proportion to the total costs. Let $\Gamma_k(X)$ denote the costs that vehicle k adds to the total costs of solution X and let $\Gamma(X)$ denote its total:

$$\Gamma(X) = \sum_{k \in K} \Gamma_k(X). \tag{5.1}$$

To introduce some randomness in the process, the vehicle of which we relocate all its trips is chosen by weighted random selection where the probability that vehicle k is selected is $\Gamma_k(X)/\Gamma(X)$. We destruct the solution by removing all trips from the selected vehicle, say k^*. The solution is repaired by iteratively assigning a trip (including the visited depot) to a different vehicle that has the lowest cost increase of doing that trip at the end of its shift. Reassigning is done in the order in which the trips were planned to be executed by vehicle k^*.

We create two versions of the destruct and repair procedure. One version of our destruct and repair procedure repeats the above only once. The advantage of doing so is that part of the solution is retained, but the disadvantage is that the perturbation of the solution may not be large enough to escape from the local optimum.

The other version repeats the procedure above the number of times the best solution has not been updated plus one. This means that if the best solution has just been updated we relocate the trips of one vehicle, because we have to disturb the solution at least by a small part before the next iteration. When vehicles have been selected for the relocation of all their trips they are no longer considered in further iterations of the current destruct and repair run. The following steps are repeated a given number of times:

1. Choose vehicle to relocate trips of.
2. Relocate all trips.
3. Do not consider chosen vehicle for remaining iterations of the current destruct and repair run.

We bound the number of iterations of this version of the destruct and repair procedure by 20% of the number of vehicles (found by preliminary testing), because otherwise the solution is disturbed too much when for a large number of iterations the best solution has not been updated.

In the classic VNS we apply the first version of our destruct and repair procedure once every 200 iterations. The reason for this low number of disturbances is that the classic VNS makes much less moves to neighbours per iteration and therefore takes more iterations before getting close to a local optimum.

5.4 Solution acceptance

In the previous sections we have discussed better or worse neighbours and increase or decrease of costs of a solution. Comparing solutions is not trivial though, because solutions may be infeasible. This forces us to develop rules for ranking solutions.

Two feasible solutions are easily ranked by their objective value. Also, a feasible solution is obviously better than an infeasible solution. When two solutions are infeasible, several different ranking methods have been used in earlier work. Penalty functions (of which many kinds are described in Coello Coello (2002, Chapter 2)) are most used, but we decide not to use those because the appropriate penalty weights for different type of violations can be very instance-specific.

Furthermore, Pareto dominance is sometimes used. This implies that a solution X_1 is better than solution X_2 if solution X_1 violates at least one constraint by a smaller value than solution X_2 and solution X_1 violates no constraints by a larger value than solution X_2. This corresponds to a decrease of at least one violation and no worsening of other violations. Although this seems attractive, we may favour a solution that has a slight increase in the violation of one constraint compared to a large decrease on other violations.

Therefore, we define a rule that accepts slightly more solutions than Pareto dominance. Of all types of constraints listed in Table 5.1, we check for the old and the new solution by what value the constraints are violated. Then, if in more constraint types there is a decline of the violation than an increase in the violation, the solution is accepted. That means that if the old solution had a violation on just one constraint type, this rule always yields the same result as Pareto dominance. If there were violations on more constraint types, all solutions that would have been accepted by Pareto dominance are also accepted with this rule, but this does not hold the other way around.

Preliminary tests show that Pareto dominance performs better than the new rule that we defined. This is probably caused by the fact that solutions are accepted when violation types increase as long as there are more violation types that decrease. The result is that the algorithm does not always move in the direction of a feasible solution. For this reason, we use Pareto dominance, with the violation types discussed above, in the experiments of the next chapter.

Type #	Violation description (quantification)
1	Minimum sourced volume (liters)
2	Maximum sourced volume (liters)
3	Compatibility between vehicles and depots (number)
4	Compatibility between depots and customers (number)
5	Compatibility between vehicles and customers (number)
6	Customer time windows (hours)
7	Vehicle time windows (hours)
8	Vehicle maximum usage time (hours)
9	Vehicle capacity (liters)

Table 5.1: List of different violation types on which we base solution acceptance.

Chapter 6

Experiments

In this chapter we discuss the experiments that we conduct to test how good the methods developed in Chapter 4 perform relative to each other and how a cluster first-route second (CFRS) approach performs compared to a one-stage optimisation approach. We also compare the different VNS heuristics that we developed. Finally, we study how the developed assignment algorithms perform when ORTEC Route Scheduling (Appendix B briefly describes the algorithms behind the software) is the routing heuristic used in the cluster first-route second approach.

6.1 Setup

The routing method that we use in the CFRS approach for the results in Section 6.2 is the first improvement VND with basic destruct and repair procedure where depots are only compatible for a customer if they have been chosen in the assignment phase. We choose the first improvement VND with basic destruct and repair because that is the basic variant from which we derived all other VNS heuristics.

We run all the experiments on an Intel® Core^{TM} i7-640M CPU. All self-written methods are programmed in C++ and to solve MILP's we make use of the open-source SYMPHONY solver of COIN-OR[1].

6.1.1 Data

The test cases that we use to test the previously discussed methods are derived from real-world cases of a global oil company. All cases consider the distribution of three products. Some general statistics that indicate the size of the cases are shown in Table 6.1. The cases vary widely in the number of orders, depots, and vehicles. This allows us to perfectly analyse the effects of problem size on relative performance of methods described in previous chapters.

[1]For more information see `https://projects.coin-or.org/SYMPHONY`.

Case no.	No. customers	No. depots	No. vehicles
1	27	5	11
2	35	5	11
3	43	11	19
4	74	11	23
5	78	11	22
6	140	9	35
7	155	9	36

Table 6.1: General statistics of test cases.

Of every described case we make three additional variants for our experiments. The base case (denoted by $x.0$ for case x) is the case without restrictions on sourced volumes, as it was supplied. In that way, we can analyse what depots are relatively much used. In the first variant (for case x referred to as $x.1$) we impose maximum sourcing volume constraints on the two depots that are used most in the base case. These depots generally have low product prices or are at a geographically attractive location. The value of the maxima are chosen by trial and error such that it does result in a change of the best solution but we are sure that a feasible solution exists. For the second variant (named $x.2$) we do the same, but then with minimum sourcing volume constraints on two depots that are least used in the base case. The third variant ($x.3$) is the most restrictive, with both maximum sourcing volume constraints on two much used depots and minimum sourcing volume constraints on two little used depots.

6.1.2 Parameters

During the experiments we will not allow customers to be assigned to more than five depots if they are not assigned to all depots. We do this because ORTEC Route Scheduling only gives the option, for every customer, to allow sourcing from all depots or list at most five depots that the customer may be supplied from. Because we want to include the effect of such a restriction, we make use of the additional procedures in Section 4.2 that enable us to work with limitations on the number of assigned depots.

Some preliminary tests, of which we do not show the results here, show that 50 iterations of the VND gives a good trade-off between solution quality and speed. We see that frequently there are still substantial improvements after 40 iterations, but running the VND for longer generally does not yield large improvements any more. Furthermore, 50 iterations approaches the upper limit on the running time that we consider acceptable for practical use (around ten minutes). The same evaluation for the classic VNS inspires us to generate the results in this chapter with 6000 iterations. Because we noted during the experiments that the running times of the VND increased tremendously with instance size we decreased the number of iterations for case 6 and 7 from 50 iterations to 10.

The average truck utilisation that we use to calculate approximated cost of assigning customers to depots is set to 1.0. We set it to this value because preliminary tests showed that during most trips the complete truck capacity is utilised.

6.1.3 Order of VNS neighbourhoods

As briefly discussed in Section 5.2, we want to do a quick check on the correctness of the order of the neighbourhoods for the VNS variants. Therefore, we run the first improvement VND with basic destruct and repair procedure with three different neighbourhood orderings. The results can be found in Table 6.2. The first column is the regular order that we hypothesise is the best one. The order of the second column has neighbourhoods 5 and 6 and neighbourhoods 8 and 9 swapped, such that smaller neighbourhoods are visited first. The order of neighbourhoods to which the objective values in the third column belong, ensures that the order of customers in a trip is optimised before possibly changing the visited depot of that trip.

It is clear from the table that the original order that we proposed gives the most consistent results. Although the original order only once gives the best result, it is also only once the worst whereas the other two orderings gave four and three times the worst results, respectively. Furthermore, the original order is the only one that finds a feasible solution in all seven base cases. Running times with the different orders are all similar, although not reported here. Because the original order give the least amount of bad results, we use that order of neighbourhoods in the experiments.

	1-2-3-4-5-6-7-8-9-10	1-2-3-4-6-5-7-9-8-10	2-3-1-4-5-6-7-8-9-10
1.0	15610	15624	15610
2.0	17439	17440	17303
3.0	27205	27065	27205
4.0	40634	infeasible	40763
5.0	37200	36908	infeasible
6.0	61613	61365	61638
7.0	68259	68647	68160

Table 6.2: Objective values of all cases without sourcing volume constraints solved with three different neighbourhood orders.

6.2 Results

For every problem instance with constraints on the sourced volume we create a routing solution by the methods listed in Table 6.3. The abbreviations shown in the last column are also used when we present the results of a method. By applying this collection of methods we can analyse many different aspects of the developed methods. We highlight the most important results in the subsections beneath. First, we compare the quality of routing solutions after single-depot assignment versus multi-depot assignment. Then, we analyse the different single-depot assignment algorithms and multi-depot assignment algorithms. After that, we discuss the performance of the different VNS heuristics. Finally, we discuss the difference in deterioration of the objective value after introducing sourcing volume constraints between the cluster first-route second (CFRS) approach and the VNS heuristics used as one-stage optimisation approach.

Single-depot assignment	Multiple depot assignment	Abbreviation
clustering + extended GAP	-	c+eGAP
clustering + extended GAP	greedy heuristic	c+eGAP+gre
extended GAP	-	eGAP
extended GAP	greedy heuristic	eGAP+gre
extended GAP	extended MKP	eGAP+eMKP
urgency heuristic (measure 1)	-	u1
urgency heuristic (measure 1)	greedy heuristic	u1+gre
urgency heuristic (measure 1)	extended MKP	u1+MKP
urgency heuristic (measure 2)	-	u2
urgency heuristic (measure 2)	greedy heuristic	u2+gre
first improvement VND with basic destruct and repair		fi VND bas
best improvement VND with basic destruct and repair		bi VND bas
first improvement VND with extended destruct and repair		fi VND ext
best improvement VND with extended destruct and repair		bi VND ext
classic VNS with first improvement local search		fi VNS
classic VNS with best improvement local search		bi VNS

Table 6.3: Different methods by which the routing solution is created for every problem instance.

Next to the main set of experiments described above, we perform a number of additional smaller experiments to analyse the following:

- How well we approximate the costs of assigning a certain customer to a specific depot and the influence of the number of customers per trip on the accuracy of the approximation.
- The influence of the number of customers per trip on the performance of the single-depot clustering heuristic.
- Comparison of the total utility found by the exact multi-depot assignment method and the greedy heuristic.
- The influence of the number of allowed depots per customer (if a customer is not assigned to all depots) on the relative performance of the exact multi-depot assignment method and the greedy heuristic.
- The influence of the number of iterations on the performance of the classic VNS.

- The influence of the number of allowed depots per customer (if a customer is not assigned to all depots) on the relative performance of the cluster first-route second approach and the one-stage optimisation.

The description of the small experiments can be found in the relevant subsections with results.

The objective values of the main set of experiments are reported in Table 6.4. The objective values (in euros) are rounded to the nearest integer. Infeasible solutions are denoted by 'inf'. In Table 6.5 we show the running times of the different methods for all the test cases. The table consists of two separate parts: the running times of the assignment algorithms when the cluster first-route second approach is used and the running times of the VNS heuristics when the one-stage optimisation is used.

For the methods that are cluster first-route second approaches we only show the running time of the assignment phase. Our routing heuristic is not optimised to run faster when a depot assignment is already made and therefore all running times are similar to the one-stage optimisation approach. Assignment running times rounded to complete seconds to overcome most irregularities that may be caused by other machine processes.

	Cluster first-route second										One-stage optimisation					
	c+eGAP		eGAP			u1			u2		VND bas		VND ext		VNS	
		gre		gre	eMKP		gre	eMKP		gre	fi	bi	fi	bi	fi	bi
1.0											15610	15574	15565	15562	15631	15631
1.1	16322	16250	16322	16250	16250	16322	16250	16250	inf	16287	16194	16091	16084	16032	16206	16206
1.2	16342	15979	16342	15979	15979	16401	15979	15979	inf	15979	16114	16104	15986	16065	16175	16175
1.3	16450	16244	16450	16244	16387	16401	16387	16387	inf	16344	16244	16097	16095	16110	16113	16113
2.0											17439	17381	17304	17344	17384	17452
2.1	18598	17726	18498	17710	17710	inf	inf	inf	inf	inf	inf	inf	17630	17613	inf	inf
2.2	18154	17544	18154	17544	17544	18197	17662	17660	inf	17486	17647	17757	17463	17402	17627	17627
2.3	18475	18243	18646	18243	18243	18890	17729	17729	inf	17750	17649	17574	17439	17488	17510	17510
3.0											27205	27194	27138	27134	27246	27246
3.1	27858	27900	27858	27900	27900	27978	27789	27789	30243	27807	27838	27807	27806	27767	28142	28142
3.2	28329	28119	28329	28119	28119	28467	28160	28160	30708	28103	28104	27951	28002	27916	28901	28901
3.3	28137	28395	28137	28395	28395	28441	28499	28542	30708	28381	28213	28077	28198	28263	28307	28307
4.0											40634	inf	41326	inf	inf	inf
4.1	41231	40926	41060	inf	inf	inf	inf	inf	inf	inf	41084	inf	41329	inf	inf	inf
4.2	inf	41931	inf	41931	41931	inf	inf	inf	inf	inf	42051	inf	41829	inf	inf	inf
4.3	inf	inf	41063	41598	inf	inf	inf	41907	inf	inf	inf	inf	41899	41856	inf	inf
5.0											37200	36767	36926	36798	inf	inf
5.1	37215	37102	37643	37090	37022	37300	37408	37592	inf	37547	37355	37344	37505	37045	inf	inf
5.2	inf	37282	inf	37417	37417	inf	37433	inf	inf	inf	37263	inf	37773	inf	inf	inf
5.3	37304	36856	37489	37069	37391	37635	37454	37905	inf	inf	37399	inf	37647	inf	inf	inf
6.0											61613	61566	61448	61238	inf	inf
6.1	inf	61867	inf	61457	61721	inf	61531	61743	inf	61573	61248	inf	61428	inf	inf	inf
6.2	inf	61897	inf	62001	62001	inf	64290	64290	inf	61917	62238	62790	inf	inf	inf	inf
6.3	inf	61892	inf	62038	62116	inf	63010	63010	inf	62326	61987	62419	inf	62131	inf	inf
7.0											68259	68981	inf	inf	inf	inf
7.1	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	71153	inf	68329	inf	inf	inf
7.2	inf	inf	inf	69050	69050	inf	inf	inf	inf	68079	inf	inf	inf	inf	inf	inf
7.3	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	69342	inf	69342	inf	inf	inf

Table 6.4: Objective value of all problem instances with all described methods.

	Cluster first-route second										One-stage optimisation					
	c+eGAP		eGAP			u1			u2		VND bas		VND ext		VNS	
		gre		gre	eMKP		gre	eMKP		gre	fi	bi	fi	bi	fi	bi
1.0											12	16	27	17	8	8
1.1	0	0	0	0	0	0	0	0	0	0	15	9	24	20	8	8
1.2	0	0	0	0	0	0	0	0	0	0	10	11	25	27	7	8
1.3	0	0	0	0	0	0	0	0	0	0	11	15	25	26	8	8
2.0											24	19	54	44	17	22
2.1	0	0	0	0	0	0	0	0	0	0	10	10	65	48	17	19
2.2	0	0	0	0	0	0	0	0	0	0	18	13	73	44	16	18
2.3	0	0	0	0	1	0	0	0	0	0	22	20	49	49	19	19
3.0											49	42	164	112	8	9
3.1	1	0	0	0	1	0	0	1	0	0	56	41	125	138	9	9
3.2	1	1	1	1	2	0	0	1	0	0	41	59	160	163	8	9
3.3	1	1	0	0	1	0	0	1	0	0	69	56	212	143	9	9
4.0											327	243	865	542	42	44
4.1	3	3	1	0	2	0	0	1	0	0	342	200	864	784	39	41
4.2	1	0	0	0	2	0	0	2	0	0	329	261	912	976	36	47
4.3	2	2	0	0	2	0	0	1	0	0	304	195	912	683	39	41
5.0											520	350	1377	934	96	103
5.1	0	0	0	0	5	0	0	2	0	0	509	424	928	848	63	66
5.2	1	1	0	0	2	0	0	2	0	0	442	242	1003	792	71	78
5.3	1	1	1	1	3	0	0	16	0	0	390	224	1035	788	64	68
6.0											1197	951	1989	1129	122	127
6.1	3	3	1	1	3	0	0	2	0	0	1728	594	1979	758	153	163
6.2	0	1	1	1	6	0	0	4	0	0	1462	1013	951	1066	111	114
6.3	1	1	3	3	8	0	0	5	0	0	1590	1088	1917	1200	137	148
7.0											2275	1748	2887	1439	195	201
7.1	1	1	0	0	4	0	0	3	0	0	2104	966	2746	1936	164	174
7.2	1	1	0	1	4	0	0	3	0	0	1808	1053	3015	1766	168	161
7.3	2	2	1	1	3	0	0	3	0	0	1838	974	1879	1343	177	182

Table 6.5: Running times of assignment algorithms for the part that is cluster first-route second and running times of the VNS variants for the part that is one-stage optimisation.

6.2.1 Single-depot versus multi-depot assignment

In the results it is clearly visible that the multi-depot assignment has a positive effect. For example, when we compare the results of 'eGAP' with 'eGAP+gre' we see that in 15 out of 21 problem instances with sourcing volume constraints the multi-depot assignment results in decreased costs. In 6 out of those 15 cases the routing heuristic could not find a feasible solution with the single-depot assignment, whereas it could with the multi-depot assignment. Over the other 9 cases the multi-depot assignment resulted in a cost decrease of 1.9% on average.

Furthermore, we see that a very bad single-depot assignment can still result in good results after the multi-depot assignment. This is particularly well illustrated by 'u2' compared with 'u2+gre'. Out of the 21 instances, routing after 'u2' only gives a feasible solution 3 times while routing after 'u2+gre' yields a feasible solution 13 times. Remarkable, and an even stronger result, is that out of those 13 times the routing solution found by 'u2+gre' is in 4 cases the best routing solution of all CFRS methods.

Despite that, we sometimes see that the routing solution found after the single-depot assignment is better than when also applying the multi-depot assignment (4 out of 21 cases when comparing 'eGAP' with 'eGAP+gre'). This perfectly shows that given an assignment, the solutions of the routing method may not always be optimal, because after the multi-depot assignment every customer can still be served from the depot to which it was assigned in the single-assignment step. Therefore, solutions after the multi-depot assignment should not be worse. When solutions with the multi-depot assignment are worse it means that the routing method got stuck in a local optimum.

6.2.2 Comparison of single-depot assignment algorithms

In this section we discuss how the various single-depot assignment methods perform in terms of objective value (after routing) and running time.

Approximation of assignment costs

Before comparing the assignment algorithms, we evaluate how well we approximate the assignment costs of assigning a specific customer to a depot (see Section 4.1.1 for the used approximation function). To this end, we create a routing solution with case 1.0, determine the costs every vehicle makes, and compare this with the approximated costs by looking up from which depot every customer is served. We use the first test case because this analysis is done manually and therefore we need to have a limited instance size. Because sourcing volume constraints are not important in this analysis we arbitrarily choose for the base case. The results are shown in Table 6.6. To analyse the effect of the number of customers per trip on the accuracy of the approximation method, we do the same analysis for case 1.0 where all order sizes have been divided by three. Those results can be found in Table 6.7.

Vehicle	No. trips	No. orders	Real costs	Approx. costs	Percentage
1	3	3	1904	1914	101
2	3	3	2031	2043	101
3	3	3	1892	1882	99
4	3	3	1769	1845	104
5	3	3	1853	1801	97
6	4	4	2376	2369	100
7	4	4	2033	1841	91
8	3	4	1848	1873	101

Table 6.6: Real and approximated costs in case 1.0 for every vehicle.

Vehicle	No. trips	No. orders	Real costs	Approx. costs	Percentage
1	2	7	1380	1358	98
2	1	3	634	593	94
3	3	10	1951	1915	98
4	3	7	1480	1436	97

Table 6.7: Real and approximated costs in case 1.0 for every vehicle when order volumes are divided by three.

The results in Table 6.6 indicate that the approximated costs are generally close to the actual costs. On average, the approximated costs of a trip are 99% of the real trip costs. This is not very surprising though, because there is only 1 trip out of 27 in which multiple customers are served. This is caused by the fact that our test cases, originating from the fuels industry, mainly contain orders of such a size that they cannot be combined with other orders. Table 6.7 shows that the number of customers per trip indeed has influence on the accuracy of the cost approximation. In that instance we never over-approximate costs and the average approximated costs as a percentage of the real costs are 97%. As described earlier (see Section 4.1.1), although the cost approximation of assigning a customer to a specific depot may generally be slightly too low if multiple customers are visited per trip, this may not be a problem as this holds for every depot.

Costs that are lower than approximated can be justified by the differing costs per hour, costs per kilometre, and vehicle capacities. For example, if a vehicle is used which is cheaper per hour and kilometre than the average costs per hour and kilometre, costs can easily be lower than approximated.

When costs are higher than approximated, this is mainly caused by the vehicle having to drive from the start location to the depot and back to the end location or because the vehicle travels a considerable time and distance between customers in the same trip.

Finally, the average truck utility may result in under-approximated as well as over-approximated costs. For example, the trips of vehicle 7 have a very low truck utilisation and this is immediately visible in the accuracy of the approximated costs. Furthermore, if a customer is served from a specific depot we assume that it returns to the same depot. When this does not hold, the costs may as well be under-approximated or over-approximated depending on the depot to which it returns.

Exact versus urgency-based assignment

From the test results we see that when we only run a single-depot assignment method the exact method outperforms the urgency-based heuristics in solving most problem instances. Especially the urgency-based heuristic with urgency measure 2 (‘u2’) performs very bad with 18 infeasible routing solutions compared to 8 with ‘eGAP’. This is probably caused by the fact that urgency measure 2 focuses very much on the degree by which minimum sourcing volume constraints are satisfied. The downside of this is that when all minimum sourcing volume constraints are satisfied the heuristic will still not start assigning customers to cheap depots.

The urgency-based heuristic with urgency measure 1 (‘u1’) performs only slightly worse than the exact method. In 10 out of 13 instances where ‘eGAP’ found a feasible solution ‘u1’ performs worse. Out of those 10 instances there are 3 instances for which a feasible solution could not be found whereas a feasible solution was found by routing after ‘eGAP’. Over the other 7 instances where ‘eGAP’ outperformed ‘u1’, the average cost decrease was just 0.6%. Routing after ‘u1’ never yields a feasible solution when ‘eGAP’ does not. The performance decrease of the urgency-based heuristic with urgency measure 1 that we expected when minimum sourcing volume constraints are included is not well visible. This may be caused, though, by the high amount of orders with large volumes due to which the procedure that fixes minimum sourcing volume constraints does not reveal its weakness.

Note that, although this did not happen in any of the problem instances of this chapter, it is possible that ‘eGAP’ finds a feasible solution whereas a heuristic approximation (including ‘u1’) does not. For ‘u1’ this happens particularly often when minimum sourcing volume constraints become more restrictive, because in the main part of the urgency-based heuristic with urgency measure 1 there is completely no focus on satisfying minimum sourcing volume constraints. When minimum sourcing volume constraints are violated after the urgency-based heuristic terminates the fixing procedure tries to solve this, but in some cases (especially with high minimum sourcing volumes) it may not be able to do this.

With respect to the running times of the exact method we see that although the problem that we solve is NP-hard, the running times are still very acceptable for test cases of real-world sizes. For only one problem instance the computation time of the exact method is more than a second. The two urgency-based heuristics never have running times of a second or longer, so this shows that in terms of speed they are a good alternative if the exact method had been slow or becomes slow for even larger instance sizes.

Effect of clustering customers

To analyse the influence of the number of customers per trip on the performance of the clustering method, we divide all order sizes of case 6 by three and run all the variants again (with minimum and maximum sourcing volume constraints adjusted to the small volumes). We do this for only one case because of the limited research time and the requirement to set new minimum and maximum sourcing volume constraints for every case that we run. The choice for case 6 is based on the fact that in the main set of experiments we see remarkable running times for 'c+eGAP'. Therefore, we would like to see what the influence of the number of customers per trip is on those running times and the associated objective values. The resulting objective values are shown in Table 6.8 and the running times (rounded to seconds) can be found in Table 6.9.

	eGAP	c+eGAP
6.1	20915	21425
6.2	21319	22826
6.3	21977	22826

Table 6.8: Objective values for cases 6.1–6.3 with order volumes divided by three.

	eGAP	c+eGAP
6.1	2	0
6.2	1	1
6.3	7	2

Table 6.9: Running times (in seconds) of the single-assignment MILP when using clusters and when using customers, for cases 6.1–6.3 with order volumes divided by three.

The clustering method is created to ensure short running times but still yield good results by taking into account more operational information than the urgency-based heuristics. The main test results indicate that clustering does still give good results (only 3 out of 21 times the results are worse than with 'eGAP' and 3 times even better than 'eGAP') but running times are not significantly lower than when not clustering customers. This may again be caused by the large order sizes. They offer very little possibility for clustering and therefore the size of the MILP's may not decrease enough to show lower running times.

In the additional experiment we see that the amount of clusters generated from 140 customers decreases from 114 (before decreasing order sizes) to 42. This shows that the order sizes were indeed preventing customers from being clustered. Table 6.9 also shows that in those cases there is a reasonable change in running times. Still, for one case the running time was the same as when not clustering. We are not sure whether this is incidental or indicates that the solver does not always recognise the simplification of the problem (in Section 4.1.3 we did not replace customers by clusters but introduced additional constraints to make sure complete clusters are assigned).

Note that the fact that larger clusters can be created when customers have smaller order sizes also has an influence on the objective values. In Table 6.8 we see that the difference between objective values is larger than in our regular problem instances. In these three instances the clustering of customers increases costs by 4.5% on average.

6.2.3 Comparison of multi-depot assignment algorithms

The comparison of the total utility found by the exact multi-depot assignment method and the greedy heuristic is reported in Table 6.10 for the three variants of case 5. In those results the single-depot assignment method is the urgency-based heuristic with urgency measure 1. We focus on comparing these values because in the main test results we see that the objective values after the greedy heuristic are lower than after the exact method for this single-depot assignment method applied to the variants of case 5.

	u1+gre	u1+eMKP
5.1	263	358
5.2	416	447
5.3	207	292

Table 6.10: Total utility of assigning customers to additional depots.

The influence of the number of allowed depots (if a customer is not assigned to all depots) on the relative performance of multi-depot assignment algorithms can be found in Table 6.11. There we compare the objective values of ‘u1+gre’ and ‘u1+eMKP’ for the variants of cases 3 and 5 when a customer may only be assigned to two depots if it is not assigned to all depots. We only do this for cases 3 and 5 because in the main test results in those particular cases ‘u1+gre’ gives better objective values than ‘u1+eMKP’.

	u1+gre	u1+eMKP
3.1	28051	28051
3.2	28160	28160
3.3	28546	28749
5.1	37376	37568
5.2	37433	inf
5.3	37439	37762

Table 6.11: Objective values of a number of problem instances for ‘u1+gre’ and ‘u1+eMKP’ when a customer may only be assigned to two depots if it is not assigned to all depots.

When we compare the objective values of ‘eGAP+gre’ with ‘eGAP+MKP’ and ‘u1+gre’ with ‘u1+MKP’ we see in the main test results that the exact multi-depot assignment method is actually not giving consistently better results than the greedy multi-depot assignment. For example, ‘eGAP+MKP’ only performs better than ‘eGAP+gre’ in 1 out of 21 cases. On the other hand, ‘eGAP+gre’ gives better results than ‘eGAP+MKP’ in 5 out of 21 cases. In 1 instance a feasible solution was found after ‘eGAP+gre’ where one could not be found after assigning with ‘eGAP+MKP’. In the other 4 cases, assignment by ‘eGAP+gre’ resulted in an average cost decrease of 0.6%. An analysis of the objective values of ‘u1+gre’ versus ‘u1+MKP’ yields approximately the same results.

Table 6.11 shows that although in cases 5.1–5.3 the results with the greedy heuristic are better than with the exact multi-depot assignment method, the total utility that the exact method finds in the multi-depot assignment phase is higher than that found by the greedy heuristic. This is as expected because otherwise at least one of our methods would be incorrect. We still see that the greedy heuristic gives better results in 5 out of 6 selected instances when we only allow a customer to be assigned to two depots if not assigned to all depots. This indicates that the number of allowed depots per customer (if not assigned to all depots) does not change the relative performance of the multi-depot assignment methods.

The running times of the extended MKP tend to be a couple of seconds for the larger cases. This is well illustrated by the running times of ‘u1+MKP’ because the urgency-based heuristic takes very little time. Remarkable is the running time of 16 seconds for problem instance 5.3. Apparently, in that particular case the solver has a lot of problems with the structure of the problem. We consider that to be a very incidental observation, though. As expected, the greedy multi-depot assignment heuristic is very quick (less than one second) for all problem instances.

The superior performance of the greedy heuristic, as discussed above, may have two causes. Firstly, it indicates that our utility calculation of assigning a certain customer to a specific depot (described in Section 4.2.1) may not correctly represent the real utility of making that additional assignment. Secondly, in the formulation of our multi-depot assignment problem, the utility of making an additional assignment does not depend on the number of depots to which a customer is already assigned. Including this, although we could not find a linear formulation for this, may ensure that an exact multi-depot assignment method gives better routing results than a heuristic.

6.2.4 Analysis of variable neighbourhood search heuristics

The first thing that we can conclude from the test results is that there is no variant of our variable neighbourhood search heuristics that always performs best. Despite that, there do appear to be some general differences for which we can try to find an explanation.

Firstly, the classic VNS performs very poorly compared to the VND heuristics. Especially for the larger cases we see that the classic VNS has more problems in finding a feasible solution, because for cases 4, 5, 6, and 7 a feasible solution is never found whereas one of the VND variants always finds a feasible solution for all but one problem instance (7.2). By looking at the running times we see that the classic VNS heuristics are always very quick. Especially for the larger cases in which they do not perform well they are a lot quicker than the VND heuristics. This is caused by the fact that the classic VNS only tries the move to one neighbour every time it visits a neighbourhood (so a constant number) whereas the VND heuristics may try all neighbours in a neighbourhood (size-dependent). The result is that for larger cases the classic VNS does not visit enough random neighbours to find one that leads to a feasible solution.

For a better comparison we would therefore increase the number of iterations of the VNS roughly by five for cases 4 and 5 and by ten for cases 6 and 7. When we do this we see that running times have increased to values that are on average equal to the running times of the first improvement VND with basic destruct and repair procedure, but the classic VNS variants still do not find feasible solutions for any of the last four test cases. Apparently, even more iterations are needed to find solutions to problems of larger sizes.

Secondly, we see that generally best improvement VND variants are faster than first improvement, but also give worse results. For the variants with basic destruct and repair procedures, first improvement yields 4 infeasible problems versus 11 infeasible problems when using best improvement. Apparently, despite the fact that in every neighbourhood all neighbours are evaluated in the best improvement variants, this is faster as less moves have to be made to achieve a certain amount of improvement. That best improvement variants give worse results has been observed by others. Hansen and Mladenović (2006) did more research on this for the case of solving a TSP with 2-opt moves. They concluded that the order of neighbours in a neighbourhood plays an important role in this phenomenon, as well as the type of neighbourhoods.

Lastly, an analysis of the difference between the basic and the extended destruct and repair procedure shows that the extended destruct and repair procedure increases running times considerably (running time of 'fi VND bas' is on average 53% of running time of 'fi VND ext'). The higher running times are a logical result of the more disrupted solution after running the extended destruct and repair procedure. The extended destruct and repair procedure results in slightly better solutions, but in some cases it has a large negative effect on the costs or results in an infeasible solution. Out of 28 cases 'fi VND ext' performs better than 'fi VND bas' in 17 cases with an average cost decrease of 0.8% if 'fi VND bas' found a feasible solution (15 cases). On the other hand, 'fi VND bas' found a better solution than 'fi VND ext' in 9 cases, consisting of 3 cases where 'fi VND ext' could not find a feasible solution. In the other 6 cases the cost decrease was on average 0.8%. This behaviour is as expected and shows that although disrupting the solution more can give better results, it can also cause too much disruption to be able to find good solutions.

6.2.5 Deterioration of solution with sourcing volume constraints

We can simply combine two previously shown tables to analyse the influence of the number of depots that a customer can be assigned to if it is not assigned to all depots on the performance of the cluster first-route second (CFRS) approach. Table 6.12 shows the objective values of a CFRS approach with 'u1+gre'. In the left column a customer is assigned to at most two depots if not assigned to all depots and in the right column a customer is assigned to at most five depots if not assigned to all depots.

	2 depots	5 depots
3.1	28051	27789
3.2	28160	28160
3.3	28546	28499
5.1	37376	37408
5.2	37433	37433
5.3	37439	37454

Table 6.12: Objective values of CFRS with 'u1+gre' when a customer is assigned to at most two versus five depots when not assigned to all depots.

During this analysis we will compare 'eGAP+gre' with 'fi VND bas'. We choose for 'eGAP+gre' because it finds a feasible solution for 18 out of 21 problem instances and is the best CFRS solution 6 times. The reason for using 'fi VND bas' in the comparison is that of all VNS variants it provides the best trade-off between solution quality and running times and finds feasible solutions most often.

Of the 21 cases with sourcing volume constraints the one-stage optimisation approach performs better than the CFRS approach 11 times. Included are the 3 cases where the CFRS could not find a feasible solution and the one-stage approach did. In the other 8 cases an average cost decrease of 0.7% was achieved by using the one-stage approach.

On the other hand, it is noteworthy that in 9 instances the CFRS approach performed better than the one-stage optimisation. This includes 3 problem instances for which the one-stage approach could not find a feasible solution and 6 problem instances in which the CFRS approach found a solution that is 0.6% cheaper on average. That the CFRS approach outperforms the one-stage optimisation in those cases is surprising because the one-stage optimisation should be able to find any solution that the CFRS approach finds. That the one-stage optimisation procedure does not find a solution equal to or better than the CFRS approach is probably caused by the one-stage procedure getting stuck in a local optimum.

Furthermore, from Table 6.12 we can see that only a small deterioration of the objective value is visible when decreasing the number of allowed depots that a customer can be assigned to if not assigned to all depots. The objective value with at most two depots is on average 100.16% of the objective value with at most five depots. In 2 of the 6 cases there is even a slight improvement in objective value when customers can be only be assigned to at most 2 depots instead of 5 (when not assigned to all depots). Again, this unexpected behaviour is probably caused by the routing heuristic getting stuck in a local optimum when customers can be assigned to at most 5 depots if they are not assigned to all depots.

6.3 Routing with ORTEC Route Scheduling

When we use ORTEC Route Scheduling as routing heuristic in the cluster first-route second approach, we get the results shown in Table 6.14. If ORTEC Route Scheduling cannot find a feasible solution it serves as many customers as possible without violating any constraints. The number of customers that can be served can be found between parentheses if no feasible solution was found where all customers are served.

A comparison of the running times of ORTEC Route Scheduling and the VNS ('fi VND bas') for solving base cases can be found in Table 6.13. We only supply the running times for base cases because we just want to give an indication of the relative speed.

	ORTEC RS	VNS	Fraction VNS/ORTEC RS
1.0	9	12	1.3
2.0	11	24	2.2
3.0	19	49	2.6
4.0	60	327	5.5
5.0	47	520	11.1
6.0	203	1197	5.9
7.0	324	2275	7.0

Table 6.13: Running times (in seconds) of ORTEC Route Scheduling (ORTEC RS) and VNS ('fi VND bas') when solving base cases.

The results in Table 6.14 (page 52) show similar relative performance of single-depot and multi-depot assignment methods to that highlighted in the various subsections of Section 6.2. When we roughly compare the objective values of ORTEC Route Scheduling and the VNS ('fi VND bas') as routing heuristic in the cluster first-route second approach, we see that the VNS is competitive with our installation of ORTEC Route Scheduling. The VNS finds slightly more feasible solutions (131 versus 116). Furthermore, when both routing heuristics found a feasible solution, the VNS found lower costs in 84 cases (average cost decrease of 0.9%). On the other hand, in 28 other cases ORTEC Route Scheduling found lower costs (average cost decrease 0.8%).

That objective values of the VNS are slightly better than ORTEC Route Scheduling is probably caused by the fact that our installation of ORTEC Route

Scheduling is not tuned to the structure of the problem instances (as hypothesised in the description in Appendix B). Comparing the running times of solving the base cases, as shown in Table 6.13, also shows that ORTEC Route Scheduling is on average five times quicker than the VNS.

	Cluster first-route second									
	c+eGAP		eGAP			u1			u2	
		gre		gre	eMKP		gre	eMKP		gre
1.1	16511	16194	16511	16194	16194	16515	16190	16190	inf (21)	16125
1.2	16524	16065	16524	16065	16065	16593	16065	16065	inf (22)	16065
1.3	16598	16321	16598	16321	16262	16593	16262	16262	inf (21)	16277
2.1	18738	17642	18626	17701	17701	inf (33)	inf (33)	inf (33)	inf (26)	inf (31)
2.2	18487	17618	18487	17618	17618	18506	17645	17567	inf (30)	17511
2.3	18589	17596	18669	17596	17596	18963	17687	17808	inf (28)	17797
3.1	28105	28135	28105	28135	28135	28213	28323	28323	inf (42)	28092
3.2	28342	28633	28342	28633	28633	28458	28530	28530	31088	28325
3.3	28551	28539	28551	28539	28539	28784	28429	28506	31088	28826
4.1	inf (73)	inf (71)	inf (72)	inf (73)	inf (73)	inf (72)	inf (71)	inf (71)	inf (60)	inf (71)
4.2	inf (62)	inf (71)	inf (63)	inf (71)	inf (71)	inf (62)	inf (72)	inf (72)	inf (62)	inf (71)
4.3	inf (72)	41452	inf (70)	inf (72)	inf (73)	inf (70)	41350	inf (69)	inf (55)	inf (71)
5.1	inf (75)	inf (77)	37675	37227	inf (76)	inf (74)	37676	38033	inf (67)	inf (77)
5.2	inf (66)	37709	inf (65)	37366	37366	inf (65)	inf (76)	inf (76)	inf (65)	37322
5.3	37551	37612	inf (77)	37314	inf (77)	inf (76)	37616	38140	inf (65)	inf (77)
6.1	inf (132)	62387	inf (133)	61829	61903	inf (130)	62005	62267	inf (130)	62247
6.2	inf (130)	63012	inf (131)	62789	62789	inf (131)	63173	63173	inf (131)	62579
6.3	inf (132)	62653	inf (134)	63026	63081	inf (131)	63275	63659	inf (127)	63156
7.1	inf (147)	inf (151)	inf (146)	inf (149)	inf (149)	inf (140)	inf (151)	inf (151)	inf (142)	inf (152)
7.2	inf (143)	68311	inf (139)	68375	68375	inf (136)	inf (149)	inf (149)	inf (136)	68529
7.3	inf (146)	inf (146)	inf (147)	inf (152)	inf (152)	inf (144)	inf (152)	inf (151)	inf (142)	inf (152)

Table 6.14: Objective value of all problem instances with ORTEC Route Scheduling as routing heuristic.

Chapter 7

Conclusions

In this chapter we discuss the conclusions that we can draw from our thesis and describe what can be focussed on in future research. We also give a recommendation to ORTEC about the methods to employ in their routing software.

7.1 Main findings

In this thesis we have come up with methods that enable us to introduce sourcing volume constraints without adapting a multi-depot vehicle routing heuristic. To leave as much possibilities as possible open for the routing heuristic, we have developed a two-stage assignment process in Chapter 4. The first stage minimises the total approximated cost of assigning every customer to one depot. The second stage maximises the total utility of assigning customers to additional depots. The results show that the addition of the second stage in the process indeed has a positive effect on the routing schedules, because the results with the second stage included are better in 15 out of 21 cases that we looked at. In 6 of those 15 cases the multi-depot assignment stage even made the difference between finding a feasible solution by cluster first-route second or not.

7.1.1 Single-depot assignment stage

To minimise the total approximated cost of assigning every customer to one depot, we have created an exact method that solves a MILP and three heuristics. Two of the heuristics are urgency-based procedures with different measures of urgency and the other heuristic clusters customers before solving the MILP of the exact method. We can conclude from the results that for the quality of the solutions of the urgency-based heuristic the chosen urgency measure is very important. The urgency measure that measures the urgency of assigning a customer to its cheapest depot relative to an average cost depot performs much better than the measure that looks at the urgency of assigning a customer to the depot that has the least satisfied minimum sourcing volume constraint for the product of which the customer has most demand relative to its cheapest depot.

Still, routing with the results of the exact method gives a better result than routing with the results of the urgency-based heuristic with the best urgency measure in 10 out of 13 instances where a feasible solution was found after assignment with the exact method. A feasible solution was never found after assignment by the urgency-based heuristic when one was not found after assignment with the exact method.

The clustering method approximately gives equivalent results for our problem instances but does not consistently run quicker than the exact method for our problem instances. Further research on the clustering method is needed to effectively use it in practice, because in instances where clustering yields quicker solutions than the exact method we also see a sharp increase in objective values. The additional research needed includes evaluation of different clustering criteria and how they perform on problem instances that have smaller order sizes. For example, we have included in the third merging criterion that customer windows must be such that customers can be helped subsequently, but we do not check whether there is a vehicle that has enough working time to subsequently serve the customers.

Critical in the analysis of our single-depot assignment procedures, is that for our problem instances we do not see running times of the exact method increase so much that the NP-hard problem could not be solved to optimality for practical use.

The approximated cost of assigning a customer to a depot that we use in the heuristics is simple but accurate in our problem instances. Despite that, we understand that we have made many assumptions in the approximation while accuracy of the values has influence on both assignment stages. Therefore, when our methods are applied to another industry (especially if many customers are visited per trip) we recommend to verify the accuracy and possibly extend the calculation to include more cost components.

7.1.2 Multi-depot assignment stage

For the maximisation of the total utility of assigning customers to additional depots, we have developed an exact method that solves a MILP and a heuristic that orders pairs of customers and depots based on the utility of assigning them and does the assignment of pairs in a greedy manner. From the test results we conclude that the heuristic yields better results than the exact method. For example, when assigning every customer to one depot by means of the exact single-depot assignment method, the greedy heuristic outperforms the exact method 5 times whereas the exact method outperforms the greedy heuristic only 1 time. Furthermore, running times of the exact method clearly grow as the problem size increases (up to a couple of seconds for our largest cases) whereas the greedy heuristic practically terminates instantly.

The explanation for the remarkable relative performance can possibly be found in two places. The first reason is that the the utility of assigning a certain customer to a specific depot may need to be calculated differently. After all, with the exact method we find a much higher total utility than the heuristic but this is not reflected in the quality of the routing schedules. Secondly, the multi-depot assignment problem that we solve to assign customers to additional depots does not take into account the number of depots a customer is assigned to. Logically, when one customer is assigned to only one depot and the other is already assigned to a number of depots it is more interesting to assign the former customer to additional depots than the latter. That we do not include this consideration in our problem may result in solutions with a lower total utility finding cheaper routing schedules. To draw any conclusions on what exactly causes the remarkable behaviour further research is needed especially on the two above mentioned reasons.

7.1.3 Variable neighbourhood search heuristics

We have constructed six variants of variable neighbourhood search heuristics to judge how first assigning customers to depots and then solving the vehicle routing problem performs relative to an approach that does vehicle routing with the consideration of the sourcing volume constraints integrated. We have created a procedure that focusses on finding a feasible initial solution or one that can be found by a low number of moves to neighbours. Ten neighbourhoods have been defined to evaluate neighbours that are increasingly different from the current solution for enough intensification in one iteration. Next to that, we have proposed a destruct and repair procedure after every iteration to offer diversification in the optimisation procedure. The variants differ in three aspects:

1. Drawing one random neighbour for every neighbourhood explored (classic VNS) or possibly exploring a complete neighbourhood before moving to the next neighbourhood (VND).

2. Choosing the first improvement found or exploring a complete neighbourhood and choosing the best improvement.

3. Destructing a fixed share of the solution after every iteration or destructing an increasing share depending on how long no improving solution has been found.

Firstly, the classic VNS does not work well for large instances. Apparently, the intensification in the classic VNS is not good enough to come to a feasible solution before the destruct and repair procedure disrupts the solution. Secondly, VND variants that move to the first improving neighbour found in a neighbourhood generally take longer but find more feasible solutions than the variants that evaluate all neighbours in a neighbourhood and then move to the best neighbour (if it is an improvement). Especially the relative quality of the solutions is remarkable, but as we shortly discussed in Section 6.2 this is probably caused by both the type of neighbourhoods and the order of the neighbours within a neighbourhood. Lastly, the extended destruct and repair procedure results in longer running times because it destructs a larger part of the solution and therefore more moves to neighbours will be made within iterations. As hypothesised, the extended destruct and repair procedure also results in cheaper solutions in 17 cases (average cost decrease 0.8%) but with the basic destruct and repair procedure a feasible solution was found in 26 out of 28 cases whereas the extended destruct and repair procedure found a feasible solution in 24 out of 28 cases. This is probably caused by the destruction of a too large share of the solution.

Generally, we see that our variable neighbourhood search heuristics perform fairly well compared to ORTEC Route Scheduling when it is not tuned for industry-specific characteristics. The objective values are competitive although running times of our VNS heuristics are much larger. When comparing the running times of our VNS heuristics with a professional software package, we of course have to keep in mind a certain factor by which our methods can run faster if programmed more efficiently.

7.1.4 Performance of cluster first-route second approach

Surprising is how well a cluster first-route second (CFRS) approach performs compared to a one-stage optimisation approach. In a CFRS approach the routing possibilities are restricted by the choices made in the phase that assigns a selected set of customers to depots. In our results we see that in 11 out of 21 problem instances our VND heuristic as one-stage optimisation procedure performs better than CFRS, but in 9 out of 21 problem instances CFRS gives better results than using our VND heuristics as a one-stage optimisation procedure.

On one hand, the superior performance of the CFRS approach in almost half the problem instances can be justified by the fact that we use heuristics to do vehicle routing. When one only uses exact approaches that find the optimal solution, it is impossible that the CFRS approach finds better solutions. On the other hand, we can also argue that the competitiveness of the CFRS approach shows that our assignment algorithms make good choices and highlight interesting regions of the total feasible region that the one-stage optimisation procedure has to explore.

7.2 Advice to ORTEC

With the results of this thesis we can assure ORTEC that their software will still be as competitive as it is now when creating the ability to introduce sourcing volume constraints by a cluster first-route second approach. Our assignment methods make sure that in many cases feasible solutions of high quality will still be found and alternative methods with guaranteed short running times are available. We especially recommend using the exact single-depot assignment method in combination with the greedy multi-depot assignment heuristic. Optionally, one can run the urgency-based single-depot assignment heuristic if the running time of the exact single-depot method exceeds a certain threshold. This ensures that the complete assignment approach is applicable in a real-world environment.

Furthermore, most of the methods that we have developed can still be used if changes are made to the problem definition. This can be particularly important for ORTEC because their routing software offers many possible variants to the problem that we have described. For a list of possible complications that we could think of and consequences for the methods developed in this thesis we refer to Appendix C.

Bibliography

A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C*, 19:723–740, 2011.

D. G. Cattrysee and L. N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60:260–272, 1992.

G. Clarke and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations research*, 12(4):568–581, 1964.

C. A. Coello Coello. Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, 2002.

F. Cornillier, F. F. Boctor, G. Laporte, and J. Renaud. An Exact Algorithm for the Petrol Station Replenishment Problem. *The Journal of the Operational Research Society*, 59(5):607–615, 2008.

F. Cornillier, G. Laporte, F. F. Boctor, and J. Renaud. The petrol station replenishment problem with time windows. *Computers & Operations Research*, 36:919–935, 2009.

F. Cornillier, F. F. Boctor, and J. Renaud. Heuristics for the multi-depot petrol station replenishment problem with time windows. *European Journal of Operational Research*, 220:361–369, 2012.

G. Croes. A Method for Solving Traveling-Salesman Problems. *Operations Research*, 6(6):791–812, 1958.

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

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

M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove. A Multiplier Adjustment Method for the Generalized Assignment Problem. *Management Science*, 32 (9):1095–1103, 1986.

A. Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

I. D. Giosa, I. L. Tansini, and O. Viera. New Assignment Algorithms for the Multi-Depot Vehicle Routing Problem. *The Journal of the Operational Research Society*, 53(9):977–984, 2002.

P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006.

S. Lin. Computer solutions of the traveling salesman problem. *The Bell Systems Technical Journal*, 44(10):2245–2269, 1965.

N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

W. L. Ng, S. C. H. Leung, J. K. P. Lam, and S. W. Pan. Petrol Delivery Tanker Assignment and Routing: A Case Study in Hong Kong. *The Journal of the Operational Research Society*, 59(9):1191–1200, 2008.

I. Or. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking.* PhD thesis, Northwestern University, 1976.

J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

S. Salhi, A. Imran, and N. A. Wassan. The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers & Operations Research*, 2013.

I. Surjandari, A. Rachman, F. Dianawati, and R. P. Wibowo. Petrol Delivery Assignment with Multi-Product, Multi-Depot, Split Deliveries and Time Windows. *International Journal of Modeling and Optimization*, 1(5):375–379, 2011.

E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31(2):170–186, 1997.

L. Tansini and O. Viera. New Measures of Proximity for the Assignment Algorithms in the MDVRPTW. *The Journal of the Operational Research Society*, 57(3):241–249, 2006.

P. Toth and D. Vigo, editors. *The Vehicle Routing Problem.* Siam, 2001.

Y. Xu and W. Jiang. An Improved Variable Neighborhood Search Algorithm for Multi Depot Heterogeneous Vehicle Routing Problem based on Hybrid Operators. *International Journal of Control and Automation*, 7(3):299–316, 2014.

Y. Xu, L. Wang, and Y Yang. A New Variable Neighborhood Search Algorithm for the Multi Depot Heterogenous Vehicle Routing Problem with Time Windows. *Electronic Notes in Discrete Mathematics*, 39:289–296, 2012.

Appendix A

Demand windows

In some applications customers define a demand window $[d^-_{np}, d^+_{np}]$ instead of a fixed demand d_{np}, for every product $p \in P$. As the total delivered volume is not part of the objective function and supplying more leads to longer shifts, theoretically there is no cost incentive to deviate from the minimum demand volume d^-_{np} for any customer and product.

On the other hand, it is beneficial to deliver a higher volume because the next delivery can then be further delayed. Therefore, after the routing method terminates it increases the delivered volumes as much as possible without violating any time window or vehicle capacity constraints. This makes the assignment phase more complicated as in that phase we do not yet know what the delivered volume will be, while that is critical to know if sourcing volume constraints are satisfied.

A.1 Strategies

How we handle the demand windows has a large influence on the quality of the solution that we get. When assigning customers to depots we can deal with the demand windows in two different ways.

The first strategy is to assign customers to depots without narrowing the demand window $[d^-_{np}, d^+_{np}]$ of any product $p \in P$ for any customer $n \in N$. Then, having found an assignment solution that satisfies the supply windows $[q^-_{mp}, q^+_{mp}]$ for every product $p \in P$ of every depot $m \in M$, the routing heuristic is given the maximum possible freedom with respect to choosing the actual delivered volume. Although this approach seems attractive, the demand windows may reduce the number of possible assignment solutions so much that a very bad assignment is chosen.

The second strategy is to narrow the demand windows (either slightly or by choosing exact volumes to deliver). The result will be that more assignment solutions are feasible and a cheaper one may be chosen. The downside of this strategy is that the routing heuristic will have to work with the narrower demand windows.

To illustrate the difference, we make use of Figure A.1. There is only one feasible solution if we want to create an assignment such that supplied volumes are within the required ranges $[q_1^-, q_1^+]$ and $[q_2^-, q_2^+]$ if the maximum possible volume is supplied to all customers. That solution is to assign customers 1 and 4 to depot 1, and customers 2 and 3 to depot 2. If the distance in the figure represents the assignment cost, this is one of the worst assignments possible. On the other hand, many more assignment solutions are feasible if we only require that the minimum demand values are within the required supply range. The optimal solution of assigning customers 1 and 2 to depot 1 and customers 3 and 4 to depot 2 is then also possible. In any case where we do not narrow demand windows, almost all solutions will be infeasible because a solution will either not be feasible if minimum demand volumes are supplied to all customers or a solution will not be feasible if maximum demand volumes are supplied to all customers. For this reason we propose to apply the second strategy and narrow down demand windows.

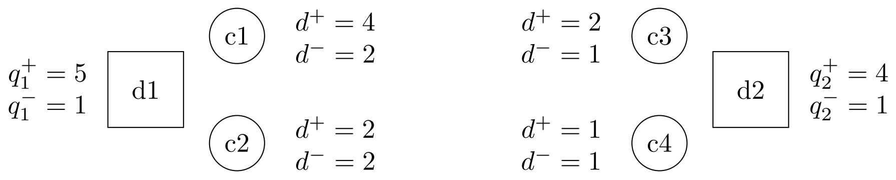

Figure A.1: Example of how narrowing the demand windows can result in a much more efficient assignment solution.

When we narrow down the demand windows to get cheaper solutions in the assignment phase, there are two more decisions to make. First we have to decide on the fixed demand value that we will temporarily assume to hold for every customer and product. Secondly, we have to decide on a way to narrow down the demand windows.

A.2 Temporarily assumed demand

In the discussed example of Figure A.1 the best assignment solution is feasible if we temporarily assume that every customer is supplied the minimum demand volume, but not if we assume that every customer is supplied the maximum allowed demand volume. This is caused by the fact that in the example of Figure A.1 the maximum supply quantities of depots are the most restricting aspects. For this reason we recommend to do the assignment based on minimum demand volumes in applications where the maximum sourcing volume constraints of depots are very tight. The other way around, in applications where the minimum supply quantities of depots heavily restrict the possibilities, we recommend to do the assignment based on maximum demand volumes.

When running times are not an issue it is of course best to apply both methods and choose the cheapest solution overall. This is also an option when it is unclear to the user what constraints are most restricting.

It may be possible that neither using d^-_{np} for all $n \in N$, $p \in P$, nor d^+_{np} for all $n \in N$, $p \in P$ will yield feasible solutions, while there may be feasible solution when choosing values between d^-_{np} and d^+_{np} for all $n \in N$, $p \in P$. In such cases one can solve a decision problem to find out if a problem is feasible at all.

A problem is proved to be infeasible if there does not exist a solution to the following framed decision problem. All parameters and decision variables other than d_{mnp} have been described in Section 4.1.1 on page 11. Decision variable d_{mnp} indicates what volume of product p is delivered to customer n from depot m, for all $m \in M$, $n \in N$, $p \in P$.

Are there values of x_{mn} for all $m \in M$, $n \in N$ and d_{mnp} for all $m \in M$, $n \in N$, $p \in P$ such that

$$\begin{aligned}
& d^-_{np} x_{mn} \leq d_{mnp} \leq d^+_{np} x_{mn}, && \forall m \in M, p \in P, && \text{(A.1)}\\
& q^-_{mp} \leq \sum_{n \in N} d_{mnp} \leq q^+_{mp}, && \forall m \in M, p \in P, && \text{(A.2)}\\
& \sum_{m \in M} x_{mn} = 1, && \forall n \in N, && \text{(A.3)}\\
& x_{mn} \leq e^{\text{dc}}_{mn}, && \forall m \in M, n \in N, && \text{(A.4)}\\
& x_{mn} \in \{0, 1\}, && \forall m \in M, n \in N. && \text{(A.5)}
\end{aligned}$$

Constraints (A.1) ensure that $d_{mnp} = 0$ if $x_{mn} = 0$ and $d^-_{np} \leq d_{mnp} \leq d^+_{np}$ otherwise. Set (A.2) forces the total supplied volume of every product at every depot to be between the given minimum and maximum values. Constraint sets (A.3)–(A.5) make sure that every customer is assigned to one depot that is compatible with that customer.

We expect that finding a feasible solution is no problem in practical cases as the minimum and maximum sourcing volume constraints will not both be very restricting. Therefore, we only develop algorithms to narrow the demand windows when either the extreme minimum or extreme maximum demand has temporarily been assumed.

A.3 Demand window narrowing algorithms

In this section we will assume that we are doing assignment based on minimum demand volumes. After having found an assignment solution with the assumption that minimum demand will be supplied to all customers, the demand windows may have to be narrowed down to make sure that the routing heuristic does not generate solutions where maximum sourcing capacities are exceeded. To this end we develop two examples of ways to do this. Algorithms 7 and 8 show these examples. On line 5 of both algorithms we sort the assigned customers based on non-increasing difference between the maximum demand volume and minimum demand volume, because we first want to narrow down demand windows of customers with wide windows.

Algorithm 7 Narrowing widest windows first.

```
Require: Feasible assignment solution based on assumption of minimum demand volume delivered to every customer.
Ensure: New maximum demand volumes $d_{np}^{+,\text{new}}$ $\forall n \in N$ to ensure feasible routing solution.
 1: $d_{np}^{+,\text{new}} = d_{np}^{+}$ for all $n \in N$
 2: for all $m \in M$ do
 3:   for all $p \in P$ do
 4:     $N_m = \{n \in N | x_{mn} = 1\}$
 5:     sort $N_m$ based on non-increasing values of $d_{np}^{+} - d_{np}^{-}$
 6:     while $\sum_{n \in N_m} d_{np}^{+,\text{new}} > q_{mp}^{+}$ do
 7:       select top customer $n_c$ from list $N_m$
 8:       set $d_{np}^{+,\text{new}} = \max(d_{np}^{-}, q_{mp}^{+} - \sum_{n \in N_m \setminus n_c} d_{np}^{+,\text{new}})$
 9:       put $n_c$ on bottom of $N_m$
10:     end while
11:   end for
12: end for
```

Algorithms 7 and 8 differ in that Algorithm 7 narrows individual customer demand windows to a fixed volume one by one, whereas Algorithm 8 gradually shrinks the demand windows of all customers simultaneously. Both algorithms will never shrink demand windows more than needed. In Algorithm 7 this is ensured by not setting $d_{np}^{+,\text{new}} = d_{np}^{-}$ on line 8 if a higher value already makes sure maximum supply volumes are never violated. In Algorithm 8 the first condition on line 9 ensures that decreasing demand window sizes never continues when it is already impossible to violate maximum supply volumes. Trivially, the described algorithms can easily be adapted to work for cases where not maximum demand volumes have to be decreased but minimum demand volumes have to be increased.

Algorithm 8 Narrowing demand windows equally.

Require: Feasible assignment solution based on assumption of minimum demand volume delivered to every customer.

Ensure: New maximum demand volumes $d_{np}^{+,\text{new}}$ $\forall n \in N$ to ensure feasible routing solution.

```
1: $d_{np}^{+,\text{new}} = d_{np}^{+}$ for all $n \in N$
2: for all $m \in M$ do
3:   for all $p \in P$ do
4:     $N_m = \{n \in N | x_{mn} = 1\}$
5:     sort $N_m$ based on non-increasing values of $d_{np}^{+} - d_{np}^{-}$
6:     while $\sum_{n \in N_m} d_{np}^{+,\text{new}} > q_{mp}^{+}$ do
7:       index $i = 1$
8:       select $i$-th customer from $N_m$
9:       while $\sum_{n \in N_m} d_{np}^{+,\text{new}} > q_{mp}^{+}$ and customer has been selected and $d_{np}^{+,\text{new}} - d_{np}^{-} > 0$ do
10:        $d_{np}^{+,\text{new}} = d_{np}^{+,\text{new}} - 1$
11:        $i = i + 1$
12:        if $i > \text{size}(N_m)$ then
13:          no customer can be selected
14:        else
15:          select $i$-th customer from $N_m$
16:        end if
17:      end while
18:    end while
19:  end for
20: end for
```

Appendix B

ORTEC Route Scheduling

ORTEC Route Scheduling[1] is the routing package of ORTEC that is used for vehicle routing problems in the oil, gas and chemicals industry. The routing method consists of a part that creates a basic feasible solution and a part that contains a set of optimisation routines.

The basic feasible solution is built by a construction heuristic. Although the package allows for the use of a couple of different construction heuristics we use the well-known insertion heuristic. In every iteration it chooses the most 'difficult-to-plan' unassigned customer (based on for example the customer's location) as the seed of the new trip. Then it orders all customers based on non-decreasing distance from the seed and places them one-by-one at the best location in the trip, until the trip is full.

The second part of the routing method contains a set of optimisation routines through which is cycled. The optimisation routines includes local search through a number of different neighbourhoods, a tabu search heuristic, and several other algorithms. The user can choose which routines have to be included in a cycle and for how long the algorithm may run. Because there are so many ways to configure the optimiser, it is possible to carefully tune it to company-specific problems. This can lead to large improvements in both solution quality and running time compared to the default configuration.

Because tuning of the optimiser settings is time-intensive and requires the knowledge of experienced implementation consultants we use a configuration that only slightly deviates from the default configuration. When evaluating the results of the experiments, we therefore do not focus on a straightforward comparison of objective values between the cluster first-route second approach and the variable neighbourhood search heuristics, but are more interested in the degradation of the objective values when introducing sourcing volume constraints.

[1]For general information visit the page with ORTEC Transport solutions at `http://www.ortec.com/`.

Appendix C

Variants to problem definition

In the table below we show a list of possible ways that we could think of to make the vehicle routing problem that we have considered more complicated. For every such complication we show the respective consequence for the methods that we have developed in Chapter 4. Every complication would require a change to the variable neighbourhood search heuristics developed in Chapter 5.

Complication	Consequence
Begin and end location of a vehicle are different.	No consequence for the developed methods.
Next to order line volumes and volume capacity on compartments, the weight has to be taken into account.	Only invalidates the third merging criterion of the clustering method (Section 4.1.3).
Besides a capacity per compartment every vehicle also has a total capacity that may be lower than the sum of the compartments.	The approximation of the assignment costs may have to be changed because it includes average truck capacity. Furthermore, again it invalidates the third merging criterion of the clustering method.
A vehicle can load at multiple depots per trip or load again before it is empty.	A detailed description of how this works with the compatibility between customers and depots is needed to know the consequences.
A vehicle can already be loaded at the start of a shift (preloaded).	Can be modelled by a depot located at the start location of the vehicle with a minimum and maximum sourcing constraint of the volume in the vehicle. That vehicle has to be the only vehicle that is compatible with the virtual depot.

A vehicle has to load for the next shift.	Can be modelled by an order with wanted product volumes located at the end location of the vehicle.
Orders are already manually planned on a trip before the optimisation is executed.	In the assignment phase the orders can be set to only be compatible with the depot that they are being served from in the manually created schedule.
Overtime is allowed.	No consequence for the developed methods.
Loading or unloading times are composed of components that have not been discussed.	The approximation of the assignment costs has to be changed. Furthermore, this invalidates the third merging criterion of the clustering method.
Complete unloading activity should be between start and end of time window.	Only affects the third merging criterion of the clustering method.
Costs include components that are not discussed.	Because of the large variety of possibilities for cost settings we cannot define a common consequence. Generally, it would be preferred to include the additional cost components in the approximation of assignment costs, but for fixed costs for the use of a vehicle this is for example not possible. This is no problem as long as those costs are not a big part of the total solution costs. If they are a big part of total costs and are not included in the cost approximation, the assignment methods will overvalue the importance of a difference in other cost components (e.g. product prices at depots).